Abstract

GANDHI, SHRUTI. Vritti: Guarantees for Mix Flows in Inter-Datacenter Networks. (Under the direction of Ioannis Viniotis.)

Inter-datacenter networks connect the geo-distributed datacenters of cloud provider(s), and many times also encompass the internet to provide connectivity. The nature of traffic on these networks is dependent on the cloud applications running in these datacenters. Hence the traffic is generally a mix of flows with and without deadlines, with known and unknown volume. The underlying wide area network (WAN) used to transfer these mix-flows is an expensive and congested resource. The providers today do not provide any guarantees to the traffic traversing the inter-datacenter WAN. In the literature, there has been little work done in guaranteeing deadlines to the traffic traversing the inter-datacenter WAN, while there has been no work done to provide tailor-made guarantees depending on the nature of traffic requirement while ensuring fairness among different flow types served in the inter-datacenter WAN environment.

In this work, we propose the problem of identifying admission control, scheduling and routing decisions to provide deadline guarantees and fairness to mix-flows in an inter-datacenter WAN environment. We use linear programming (LP) to mathematically formulate the problem with the objective of maximizing utility that is a function of revenue generating ability of the flow types and fairness among all flow types. We propose a spatial-temporal traffic engineering system ‘Vritti’, that can provide these guarantees. We propose and solve the static version of the problem where the future requests are known to the system in advance. We then propose a more realistic dynamic problem where the future requests are not known in advance. We propose four algorithms, namely, greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair to provide guarantees to mix-flows. We evaluate the effectiveness of these algorithms using extensive simulations. With selective-rescheduling, we achieve close to 69% acceptance rate of hard deadline requests at arrival rate of 10, and close to 100% acceptance rate for lower arrival rates. Using greedy-fair algorithm, we achieve close to 50% acceptance rate for hard deadline requests and close to 100% fraction of non-deadline requests allocated at arrival rate of 10. With selective-rescheduling-fair algorithm we strike a nice balance between the aggressive deadline traffic oriented approach taken by selective-rescheduling algorithm and fairness oriented approach taken by the greedy-fair algorithm. We also propose problem variations in the context of business models such as federated-cloud and multi-cloud. We solve the problems in these models by modifying greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair algorithms and evaluate the algorithms comprehensively using simulations.
Dedication

To my parents, Vritti, Raman and Manish, who made this possible.

* * *

Saruman believes it is only great power that can hold evil in check, but that is not what I have found. I found it is the small everyday deeds of ordinary folk that keep the darkness at bay. Small acts of kindness and love.

- Gandalf, The Hobbit: An Unexpected Journey
Biography

Shruti Gandhi was born in New Delhi, India in 1991. She received her B.Tech degree in Electronics and Communication Engineering from Indira Gandhi Institute of Technology, Guru Gobind Singh Indraprastha University, India in 2013, and an MS in Computer Networking from North Carolina State University in Raleigh, NC in 2016. She worked in Power Generation and Automation group of ABB from 2013 to 2014. She worked as a software engineering intern in Routing and Optical Switching Group at Cisco in summer and fall of 2015, in Emerging Technology Institute, Hyperledger Fabric Group at IBM in summer of 2016. In summer of 2017, she worked as a PhD intern at Cisco again in the Service Provider Networking and Software Automation Group. She was the recipient of the IBM PhD Fellowship for the year 2017-18.

At NC State University, Shruti was a teaching assistant and lab instructor for the undergraduate course - ‘Internetworking’, and the graduate course ‘Internet of Things’ from 2016 to 2019. She also led the IEEE Women In Engineering group for the Eastern North Carolina Section from 2016 to 2019.

She is happily married to Manish Khilnani since 2017.
Acknowledgements

It takes a village to give rise to a doctorate, and I was no exception. If it were not for the patience, hard work and belief of my advisor, committee members, family, friends and my colleagues, I wouldn’t have been able to see the light at the end of the tunnel. So bear with me, this is going to be a long and emotional section.

* * *

First, I would like to thank my advisor, Dr. Yannis Viniotis. When I joined NC State University in Fall of 2014 as a masters student in computer networking, I took two courses taught by Dr. Viniotis - ‘Network Design and Management’ and ‘Internet of Things: A Primer’. Prior to joining the graduate school, my research background was in digital signal processing, and I planned to switch to systems and networking. Hence, it was quite a risk on Dr. Viniotis’s part when he let me co-write with him the reference text for ‘Network Design and Management’ once I had completed the course. In my third semester of masters, he took me up as a masters thesis student and in the subsequent semester, gave me my first teaching job. He encouraged me to explore varied research topics, but made sure I always progressed with a grounding in reality. He taught the most fundamental yet most profound principles of computer networking with hats (perspective of approaching a problem), knobs (control-knobs), cake (you cannot have your cake and eat it too!), and Peter and Paul (we must borrow from Peter to pay Paul)! Besides guiding me in my research for the past six years, he taught me the necessary skills to navigate life itself as a student, researcher and an engineer. For some reason, he had more belief in me than I ever had in myself, and that is what kept me going through the highs and lows of this long winding road. I have been incredibly blessed to be advised by Dr. Viniotis, a systems and networking savant who also happens to be the most kind and empathetic human being.

In addition to Dr. Viniotis, I would like to thank Dr. Perros. I took two courses with him, ‘Networking Services’ and ‘IoT Analytics’. It was from him I learnt the fundamentals of QoS, queuing theory and statistical analysis; topics that were key to my research. Dr. Perros, along with Dr. Byrd and Dr. Shahzad as my committee members provided me with invaluable guidance and feedback all these years, that helped me drive my research forward.

I would like to express my gratitude towards IBM, and Dr. A.J. Rindos for considering me for the prestigious IBM PhD fellowship, that helped me continue my research without any constraints.

In the summer of 2016, Dr. A.J. Rindos, gave me the opportunity to intern at IBM in the Emerging Technology Institute. I worked under the guidance of Kostas Christidis to develop a proof-of-concept electronic health record management system on Hyperledger Fabric. It was here
that I learnt the process of fine-tuning system design by understanding business requirements directly from the stakeholders involved. Kostas’s enthusiasm towards his work and his clarity of vision are qualities that I have aspired to since.

I interned at Cisco in summer and fall of 2015 under the guidance of Sadasiva Reddy Mopuri and Sree Rama Nomula in the Routing and Optical and Switching Group. This was my first industrial experience in the computer networking domain and I thoroughly enjoyed it, thanks to the amazing team I got to work with.

I interned in Cisco again as a PhD intern in the summer of 2017 under the guidance of Santiago Alvarez. In this internship I got the opportunity to contribute to the open source project ‘Yang Development Kit’. Santiago led the project with great dynamism that was infectious. I also got the opportunity to collaborate with two wonderful software engineers - Abhi Ramesh Keshav and Xiaoqin Zhu.

I would like to thank Dr. Anand Singh for his guidance and encouragement all through these years, and the jolly presence of a fellow PhD in making, Sudhendu Kumar, with whom I often shared the chore of grading exams (not that I consider grading exams a chore).

I would like to thank Savera Tanwir and Magreth Mushi, for taking me under their wings to lead the IEEE women in engineering group.

There were countless times during the last six years, I had to run to the ECE graduate office for one administrative reason or the other. The ever-beaming Amy Roosje and Fenile Jones were always patient, kind and helpful in answering all my queries.

* * *

Outside of ECE graduate school, my friend and housemate Indunil Angunewala who herself treads a much tougher path of pursuing PhD in Physics, formed my support system. We would often go on long walks in the evening discussing our research problems, and life in general. She was primarily the test audience for my research presentations for the qualifying and preliminary exams, and provided valuable feedback. I would also like to thank Sunita Aggarwal and Kiran Rathore, my two sisters from another mother, for their unconditional love and support, and for sorting my weekends.

Before I moved to the United States, I worked briefly in ABB in Bangalore, India. I was extremely fortunate to have worked with the most wonderful, kind-hearted people during that stint, like my manager, Shankar Raman and my mentor, Sankara Subramanian. Sankara with his affable presence, would grin and go-about the toughest of tasks. He was the best mentor, and Shankar the best manager, I could have asked for at the start of my career.

My foray into research started as an undergraduate when Dr. Akash Tayal gave me the opportunity to work with him on a digital image processing project. He pushed me to delve deep into topics which at the time were outside my area of comfort, such as linear algebra and
algorithm design. He taught me to be fearless when approaching complicated topics, a skill that proved to be crucial during my research journey.

* * *

Saving the best for the last, I would like to thank my parents, Neelam and Sanjeeva Gandhi. They did, what in hindsight I feel is the most difficult thing parents can do, to send their first born child out of their nest, to a strange country, miles and miles away just so their child can pursue her dreams. I cannot thank them enough for their faith in me, their unconditional love and support. My mother, with her empathy and indomitable spirit, and my father with his hard-work and grit, have always been my biggest sources of inspiration. They inculcated in me the importance of being independent early on in life, they encouraged me to pursue my dreams and taught me to never take any amount of success seriously. They are my beacons of light in this world and whatever I achieve in life, I owe it to them.

I would also like to thank my other set of parents, Pooja and Ghanshyam Khilnani, for their love and encouragement all throughout these years. I would also like to thank Neha Khilnani for being the cutest fuzzball of light in my life.

When I try to remember my childhood, I most vividly remember the parts I spent with my two siblings, Vritti and Raman. Vritti, who is three years younger than me in earth years, and at least five years older than me mentally, has been the only person I have ever confided everything into. Also, she is my free version of grammarly, a writing assistant software. She is the yin to my yang. I cannot imagine how my life would have turned out, if it wasn’t for her. Raman, who is the youngest of us three, was only fourteen when I came to the United States. He has grown into a mature adult now, and has many a times surprised me with his compassionate and egalitarian outlook on life. Long winded discussions with him, whether technical or about life in general, have made me review my outlook on things.

Finally, Manish Khilnani, my husband, my best friend and my biggest support system in the whole wide world. He came into my life like a ray of sunshine, he picked me up from my many spells of procrastination, and enabled me to keep working hard. He had the toughest job in the world, waiting, and wait he did, patiently, and for that alone I will be eternally grateful. I couldn’t have imagined going through this journey without him. It had its highs and lows, but the only reason I was able to go through it all, was knowing that he was beside me no matter what.
# Table of Contents

List of Tables ................................................................. x

List of Figures ................................................................. xi

**Chapter 1 Introduction** .................................................. 1
  1.1 Inter-datacenter Networking in Cloud Computing ................. 1
  1.2 Motivation ................................................................. 2
    1.2.1 The Need for Handling Mix-flows ............................ 4
    1.2.2 The Need for Tailor-made Guarantees for Transfers .......... 5
    1.2.3 The Need for Fairness in Utility Definitions ............... 5
  1.3 The Philosophy .......................................................... 5
  1.4 Contents of Dissertation .............................................. 6
    1.4.1 Challenges .......................................................... 7
  1.5 Contribution ............................................................ 7
  1.6 Outline of Dissertation ............................................... 8

**Chapter 2 Background and Literature Survey** ....................... 9
  2.1 Background ............................................................... 9
    2.1.1 Definitions .......................................................... 9
    2.1.2 Assumptions ......................................................... 13
    2.1.3 Commercial Inter-DC WANs ....................................... 14
  2.2 Literature Survey ....................................................... 14
    2.2.1 Classification by Objective and System Environment ......... 15
    2.2.2 Classification by Type of Workload .............................. 20
    2.2.3 Classification by Control Knobs ................................ 22
    2.2.4 Classification by Performance Evaluation Characteristics .. 29
    2.2.5 Summary .............................................................. 36

**Chapter 3 Vritti: Static Problem in Single Cloud** .................. 45
  3.1 Design Goals ............................................................. 45
    3.1.1 Maximizing Provider Revenue .................................... 46
    3.1.2 Providing Fairness ................................................ 47
    3.1.3 Providing a Control Knob for Revenue vs. Fairness Trade-off 47
  3.2 System Model ............................................................. 47
    3.2.1 WAN Traffic Model ................................................ 48
    3.2.2 Network Model ....................................................... 51
    3.2.3 Revenue Model ....................................................... 53
    3.2.4 Fairness Model ....................................................... 55
  3.3 Static Problem ........................................................... 56
    3.3.1 Exact Problem Statement ......................................... 57
    3.3.2 Mathematical Problem Formulation ............................... 57
    3.3.3 Control Knobs ....................................................... 60
    3.3.4 Mathematical and Business Novelty .............................. 60
Chapter 6 Conclusion

6.1 Conclusion

6.1.1 Problems Solved

6.2 Future Work

6.2.1 Theoretical Analysis of the Algorithms

6.2.2 TCP for Mix-flows in Inter-DC Networks

6.2.3 Large Scale Implementation of Algorithms for Federated and Multi-Cloud Environments

6.2.4 Point to Multi-point Resource Allocation in Inter-DC Networks
## List of Tables

Table 1.1 Cloud Consumer and Operator Interview at Microsoft [41]. .......................... 5

Table 2.1 Literature Survey w.r.t the objective met and system environment considered. In every cell, S - Single cloud, M - Multi cloud, F - Federated-cloud. 19

Table 2.2 Literature classification w.r.t the type of workload considered to meet the objectives. See Section 2.1.1 for workload definitions. ............................... 23

Table 2.3 Classification by the perspective used, SLA guaranteed, if admission control, scheduling, path selection, buffer allocation are used along with additional control knobs. ........................................ 28

Table 2.4 Classification by performance evaluation characteristics. .............................. 37

Table 2.4 Classification by performance evaluation characteristics (continued). ........... 38

Table 3.1 Tenant Transfer Request Model. ................................................................. 51

Table 3.2 Traffic Model Summary. ........................................................................ 51

Table 3.3 Notations and their definitions. .................................................................. 56

Table 3.4 Problem Size for the Static Problem. ....................................................... 59

Table 3.5 Mathematical Model Differences with Tempus, PGA, Amoeba, Pretium with Vritti. ......................................................................................... 62

Table 3.6 Business Model Differences with Tempus, PGA, Amoeba, Pretium with Vritti. ......................................................................................... 62

Table 3.7 Paper Comparison in Multiple Business Models. ..................................... 62
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A Typical Datacenter Topology</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Microsoft’s inter-datacenter network with 37 sites [52]</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Google’s inter-datacenter network with 33 sites [54]</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Vritti: System Model</td>
<td>53</td>
</tr>
<tr>
<td>3.2</td>
<td>Acceptance Rate for Type 1 and Type 3 Flows in G-Scale</td>
<td>67</td>
</tr>
<tr>
<td>3.3</td>
<td>Average Fraction Guaranteed for Type 2 and Type 3 Flows in G-Scale</td>
<td>67</td>
</tr>
<tr>
<td>3.4</td>
<td>Average Fraction Allocated for Type 4 Flows in G-Scale</td>
<td>68</td>
</tr>
<tr>
<td>3.5</td>
<td>Average Fraction Allocated for Type 5 Flows in G-Scale</td>
<td>68</td>
</tr>
<tr>
<td>3.6</td>
<td>Network Utilization in G-Scale</td>
<td>69</td>
</tr>
<tr>
<td>3.7</td>
<td>Average throughput per request in G-Scale</td>
<td>70</td>
</tr>
<tr>
<td>3.8</td>
<td>Revenue due to all Requests in G-Scale</td>
<td>71</td>
</tr>
<tr>
<td>3.9</td>
<td>Time Taken for Request Allocation in G-Scale</td>
<td>72</td>
</tr>
<tr>
<td>3.10</td>
<td>CPU Usage for Request Allocation in G-Scale</td>
<td>72</td>
</tr>
<tr>
<td>3.11</td>
<td>Memory Usage for Request Allocation in G-Scale</td>
<td>73</td>
</tr>
<tr>
<td>4.1</td>
<td>Acceptance Rate for Type 1 and Type 3 Flows in G-Scale</td>
<td>91</td>
</tr>
<tr>
<td>4.2</td>
<td>Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale</td>
<td>92</td>
</tr>
<tr>
<td>4.3</td>
<td>Average Fraction Allocated for Type 4 Flows in G-Scale</td>
<td>93</td>
</tr>
<tr>
<td>4.4</td>
<td>Average Fraction Allocated for Type 5 Flows in G-Scale</td>
<td>94</td>
</tr>
<tr>
<td>4.5</td>
<td>Average Network Utilization in G-Scale</td>
<td>95</td>
</tr>
<tr>
<td>4.6</td>
<td>Average Throughput per Request in G-Scale</td>
<td>96</td>
</tr>
<tr>
<td>4.7</td>
<td>Acceptance Rate for Type 1 and Type 3 Flows in G-Scale</td>
<td>97</td>
</tr>
<tr>
<td>4.8</td>
<td>Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale</td>
<td>97</td>
</tr>
<tr>
<td>4.9</td>
<td>Average Fraction Allocated for Type 4 Flows in G-Scale</td>
<td>98</td>
</tr>
<tr>
<td>4.10</td>
<td>Average Fraction Allocated for Type 5 Flows in G-Scale</td>
<td>98</td>
</tr>
<tr>
<td>4.11</td>
<td>Time Taken in G-Scale</td>
<td>99</td>
</tr>
<tr>
<td>4.12</td>
<td>Memory Usage in G-Scale</td>
<td>100</td>
</tr>
<tr>
<td>4.13</td>
<td>Time Revenue in G-Scale</td>
<td>101</td>
</tr>
<tr>
<td>4.14</td>
<td>Average Throughput in G-Scale using Greedy</td>
<td>102</td>
</tr>
<tr>
<td>4.15</td>
<td>Average Throughput in G-Scale using Greedy-fair</td>
<td>102</td>
</tr>
<tr>
<td>4.16</td>
<td>Average Throughput in G-Scale using Selective Rescheduling</td>
<td>103</td>
</tr>
<tr>
<td>4.17</td>
<td>Average fraction guaranteed for type 2 and 3 flows in G-Scale with greedy</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>algorithm with duration of request calculated as Exponential distribution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>with mean in [2,4,6,8,10,12].</td>
<td></td>
</tr>
<tr>
<td>4.18</td>
<td>Average fraction guaranteed for type 2 and 3 flows in G-Scale with GF</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>algorithm with duration of request calculated as Exponential distribution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>with mean in [2,4,6,8,10,12].</td>
<td></td>
</tr>
<tr>
<td>4.19</td>
<td>Average fraction guaranteed for type 2 and 3 flows in G-Scale with SR</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>algorithm with duration of request calculated as Exponential distribution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>with mean in [2,4,6,8,10,12].</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.20 Average fraction guaranteed for type 2 and 3 flows in G-Scale with SRF algorithm with duration of request calculated as Exponential distribution with mean in [2,4,6,8,10,12].

Figure 5.1 System Architecture for Federated-Cloud.

Figure 5.2 System Architecture for Multi-Cloud.

Figure 5.3 Algorithm Modifications in Federated-cloud Environment.

Figure 5.4 Acceptance Rate for Type 1 and Type 3 Flows in G-Scale.

Figure 5.5 Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale.

Figure 5.6 Average Fraction Allocated for Type 4 Flows in G-Scale.

Figure 5.7 Average Fraction Allocated for Type 5 Flows in G-Scale.

Figure 5.8 Average Network Utilization in G-Scale.

Figure 5.9 Average Throughput in G-Scale.

Figure 5.10 Time Taken in G-Scale.

Figure 5.11 Memory Usage in G-Scale.

Figure 5.12 Acceptance Rate for Hard Deadline Flows in G-Scale if Only Hard Deadline Requests in Workload.

Figure 5.13 Average Fraction Guaranteed for Soft Deadline Flows in G-Scale if Only Soft Deadline Requests in Workload.

Figure 5.14 Acceptance Rate for Type 3 Flows in G-Scale if Only Type 3 Requests in Workload.

Figure 5.15 Average Fraction Guaranteed for Type 3 Flows in G-Scale if Only Type 3 Requests in Workload.

Figure 5.16 Average Fraction Allocated for Type 4 Flows in G-Scale if Only Type 4 Requests in Workload.

Figure 5.17 Average Fraction Allocated for Type 5 Flows in G-Scale if Only Type 5 Requests in Workload.

Figure 5.18 Acceptance Rate for Type 1 and Type 3 Flows in G-Scale.

Figure 5.19 Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale.

Figure 5.20 Average Fraction Allocated for Type 4 Flows in G-Scale.

Figure 5.21 Average Fraction Allocated for Type 5 Flows in G-Scale.
Chapter 1

Introduction

1.1 Inter-datacenter Networking in Cloud Computing

There has been a steady rise in cloud computing adoption over the past few years. The consumers of the cloud can lease infrastructure, platform and software to meet their diverse workload demands. Companies such as Amazon, Microsoft and Google have built a geo-distributed network of datacenters to meet their customers’ demands, as well as to host their own parallel businesses. A datacenter typically hosts multiple inter-connected servers. The servers are stacked in racks and inter-connected in a network topology, Figure 1.1 shows a typical datacenter network topology. Each physical server in these datacenters hosts multiple virtual machines, which in most cases is the entity leased to a tenant. The capital and operational expenditure of building and maintaining a datacenter can be burdensome for individuals as well as small and large enterprises. Leasing compute, storage and network capacity from a cloud provider is hence a more cost and time effective solution. Formally, cloud computing can be defined as a service provided by enterprises called cloud providers, that allow its consumers (or tenants) to rent compute, storage and networking resources bound by a contract called the service level agreement (SLA).

A cloud computing environment can be categorized into public cloud, private cloud or a hybrid cloud. The resources in a public cloud (e.g., AWS [62], Azure [63], GCP [68], Rackspace [69]) are shared among multiple tenants or organizations. These tenants are typically charged in a pay-as-you-go fashion for their usage. In contrast, in a private cloud, the resources are not shared with any other organization, and typically there is no monetary transaction for usage. Most companies run their own private cloud which is used by their employees. Similarly, universities typically run private cloud for their faculty, students and employees. A hybrid cloud is a combination of one or more public clouds with private clouds and/or on premises
computing. For example, a mixed computing, storage, and services environment made up of on-premises infrastructure, private cloud services, and a public cloud — such as Amazon Web Services (AWS) or Microsoft Azure, with orchestration among the various platforms [61]. The work in this dissertation applies to all three types of cloud.

Large cloud providers such as Amazon AWS [62], Microsoft Azure [63], Google GCP [68], IBM [65], Alibaba [64], serve tenants at a global level and have thus deployed a globally distributed infrastructure of datacenters in order to provide lower response times, reliability, availability and better load balancing opportunities. The network connecting these geo-distributed datacenters is called the inter-datacenter wide area network.

1.2 Motivation

The inter-datacenter wide area network connecting geo-distributed datacenters is a critical and expensive resource, costing annually hundreds of millions of dollars for the cloud providers [26], [25], [57]. Google’s private, software defined, wide area network has seen an exponential growth of traffic, with bandwidth requirements growing hundred times over a period of five years from 2013 to 2018 [54]. Many services rely on low-latency inter-datacenter communication for good user experience and on high-throughput transfers for reliability (e.g., when replicating updates) [25]. An interesting attribute of these networks is that they have a limited number of end-points to connect, and have capacity in the order of 100 Gbps to 1 Tbps. In Figures 1.2 and 1.3, we can see Microsoft’s and Google’s inter-datacenter wide area networks, respectively.

Like any fast growing resource, the inter-datacenter WANs have been plagued with chal-
Figure 1.2: Microsoft’s inter-datacenter network with 37 sites [52].

...lenges. The chief among those is the lack of guarantees in general for traffic traversing these networks.

Historically, both public and private WANs were grossly over provisioned in order to meet peak traffic demands. However, with the developments in software defined networking (SDN), it has now become possible to utilize the knowledge of global network state to make optimal traffic engineering decisions. Google and Microsoft in 2013 re-architected their networks (B4 [26], SWAN [25] respectively) to leverage SDN and deploy optimal traffic engineering decisions to their network with the objective of maximizing network utilization. However, there is no cloud provider that provides guarantees to the traffic traversing their WAN i.e., there are no service level agreements (SLAs) for the bandwidth resource allocated along inter-datacenter WANs. In a recent survey done by Jalaparti et. al. [41], of the 15 operators and consumers of Microsoft WAN, the authors found that there exists no evidence of formal SLA enforcement for transfers in the inter-datacenter WANs. However, all the interviewees concurred that some form of guarantees in terms of deadlines are required for the inter-datacenter WAN transfers. While some of the recent works [29], [57], [53] in the area provide guarantees to inter-datacenter WAN transfers, there are still relevant open problems, the brunt of which is borne by the customers of the cloud providers.
1.2.1 The Need for Handling Mix-flows

Cloud applications generate a mix of flows, varying in size and guarantee requirements. To the best of our knowledge, in all the recent works the flows provided with deadline guarantees have been bulk transfers, and all the remaining traffic has been treated as background traffic. The interactive traffic is not handled by the traffic engineering solutions proposed because of strict latency requirements. However the spectrum of transfer size that is dealt with as background transfers is large, varying between small transfers and bulk transfers. The guarantees that have been proposed in recent works have been limited to strict hard deadlines, or in case of Tempus [29] only soft deadlines. Also, no work has dealt with deadline transfers in conjunction with non-deadline transfers. According to the interview conducted by Jalaparti et. al. [41] specified in Table 1.1, 40% transfers on average do not have strict deadlines. If schemes for deadline and non-deadline flows are trivially combined, they would end up prioritizing deadline flows hurting the flow completion time (FCT) for non-deadline flows [39]. If a customer has a workload with strict deadline requirement (e.g., web search request), another with a looser deadline requirement (e.g., VM migration), another with no deadline requirement (e.g., redundancy backups, database access), there has been no work that can provide guarantees to this mix of workloads without hurting the flow completion times of flows conventionally categorized as lower priority flows. It is important to remember there are customers who may suffer due to inaccurate categorization of their workloads if they were unable to accurately represent their requirements. Hence, there is a need for an exhaustive traffic classification to represent the
varied spectrum of traffic requirements and the need for a system that provides guarantees to all flow types.

<table>
<thead>
<tr>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do any transfers require deadlines?</td>
<td>100% said yes</td>
</tr>
<tr>
<td>What fraction of transfers have strict deadlines?</td>
<td>60% on average</td>
</tr>
<tr>
<td>Would you pay extra for guaranteed deadlines?</td>
<td>64% said yes</td>
</tr>
</tbody>
</table>

**1.2.2 The Need for Tailor-made Guarantees for Transfers**

The guarantee for a transfer with hard deadline must be full completion transfer by the hard deadline, else the transfer is deemed useless and bandwidth is wasted. It is not logical to guarantee a partial transfer completion by a deadline for a hard deadline transfer request. Similarly, it is useless to guarantee full transfer completion for soft deadline transfers by the soft deadline, as the bandwidth allocated could have been used for more urgent transfers. Also, it is unfair to treat non-deadline flows as background flows. Hence, there is a need to handle the mix of flows such that the most appropriate guarantee is allocated to the transfer. To the best of our knowledge, there has not been any work published where hard deadlines, soft deadlines, a combination of hard and soft deadlines, and no deadlines are considered in conjunction.

**1.2.3 The Need for Fairness in Utility Definitions**

As proven in Karuna [39] for intra-DC networks, if schemes for deadline and non-deadline flows are trivially combined, they would end up prioritizing deadline flows hurting the flow completion time (FCT) for non-deadline flows. Any such scheme would end up starving requests with minimal and no guarantees, this could affect the overall provider utility. In Tempus [29] the authors provided fairness to all transfers by guaranteeing the completion of the same fraction of tenant request for all requests. Hence, effectively all transfers are always treated the same, which is unrealistic. A better scheme would implement a mechanism to achieve trade-off between the contradictory requirements of high provider revenue and fairness among all requests.

**1.3 The Philosophy**

Our philosophy is to solve the problem in a way that the solution remains highly applicable in the face of changing customer demands and cloud provider goals, as well as evolving cloud
• **Fairness vs revenue trade-off** - As a part of the philosophy we develop solutions to the problem that clearly model the revenue versus fairness trade-off. To do so, we provide customers the means to express their demands without compromising on their requirements. And we provide a control knob to the cloud provider to make judicious decisions in allocating workloads that best meet their top level objectives.

• **Applicability to evolving cloud computing models** - To maintain applicability in the face of evolving cloud computing architectures, we model the problem in single cloud, federated-cloud and multi-cloud environments. The solution is modified to accommodate the underlying system model.

A work can be driven by any philosophy. We adopted the philosophy of adaptability because the world of cloud computing is changing fast, and any system adding value in that arena must be built for sustaining that change. While no one can predict the future, we *can* make informed design decisions based on known patterns. We step into the shoes of cloud providers and cloud customers to remain prudent in anticipating the future of the cloud computing models, customer demands and provider goals. We mold our solution to compensate for these changes. We could have chosen to maximize revenue if we wore a cloud provider hat or to maximize fairness if we wore a cloud customer’s hat. However, we chose adaptability over a static perspective, wherein we cater to people wearing either hat. This is the reason we decided to model the trade-off between two intrinsically contradictory objectives of fairness and revenue. We chose to make the solution applicable in multiple cloud computing environments, as the single cloud model is no longer sufficient when it comes to customer adoption and is fast changing to make way for multi-cloud and federated-cloud. We describe this in detail in Section 2.2.1.

### 1.4 Contents of Dissertation

The dissertation focuses on design and evaluation of Vritti, a system that provides guarantees for mix-flows in inter-datacenter wide area networks. *Vritti* solves the following three problems:

• How to provide tailor-made guarantees for mix-flows in inter-datacenter wide area networks?

• How to achieve revenue versus fairness trade-off while providing guarantees to mix-flows?

• How to modify a system designed to provide transfer guarantees to fit into federated and multi-cloud environments?
1.4.1 Challenges

For the first and the second problem, \textit{Vritti} must overcome the following challenges to solve the problems effectively:

- \textit{Vritti} must allow the customers of the cloud providers to clearly express their workload requirements. It must \textbf{exhaustively define the categorization of workload to encompass the complete spectrum of workload requirements}.
- \textit{Vritti} must be able to \textbf{solve the spatial temporal online problem} i.e., it must be able to allocate requests across space (along paths) and time (in the future, calendaring) as they dynamically arrive in the system. It must jointly optimize over all the requests without reneging on previous promises.
- \textit{Vritti} must be able to \textbf{increase overall network utilization} to improve cloud provider revenue and motivate them towards fairness friendly solution. To do so, it must use higher path diversity to allocate requests.
- \textit{Vritti} must be able to define the control knob(s) to \textbf{accurately model the trade-off of cloud provider revenue versus fairness among all request types}.

For solving the third problem effectively, \textit{Vritti} must overcome the following challenges:

- \textit{Vritti} must be able to keep its promises of providing guarantees to mix-flows irrespective of the underlying architecture.
- \textit{Vritti} must be able to \textbf{tune and/or modify the trust boundaries among different organizations} such that it can provide the guarantees it promised.

1.5 Contribution

We make the following three contributions:

1. We propose a system \textit{Vritti}, that provides guarantees for mix-flows in inter-datacenter wide area networks. We formulate the static problem solved by \textit{Vritti} as a Linear Program and evaluate its solution using simulations.

2. We propose four algorithms employed by \textit{Vritti} to solve the corresponding dynamic problem in a single cloud environment. We evaluate the algorithms using simulations.

3. We propose a system model for \textit{Vritti} in federated-cloud and multi-cloud environments. We modify the proposed algorithms employed by \textit{Vritti} to solve the dynamic problem in the single cloud environment such that they are employable in federated-cloud and multi-cloud environments. We evaluate the modified algorithms using simulations.
1.6 Outline of Dissertation

The remainder of the dissertation is divided into six chapters. In Chapter 2, we describe the inter-datacenter wide area network environment in more detail and we motivate the pursuit of the problems by Vritti using an exhaustive literature survey.

In Chapter 3, we outline the design goals and define the system model, followed by the static problem formulation. In Chapter 4, we formulate the dynamic problem, propose algorithms to solve the problem, followed by an extensive evaluation of the algorithms.

In Chapter 5, we modify the system model to accommodate Vritti in the federated-cloud and multi-cloud environments. We evaluate the solutions to the dynamic problem in these environments.

In Chapter 6, we conclude the study.
Chapter 2

Background and Literature Survey

In this chapter, we provide the background necessary to understand the dissertation, followed by an extensive literature survey of the related work that has been done in the area of inter-datacenter wide area networking. With the comprehensive survey of the research area we identify the key open areas of research, which subsequently aid in formulation of the problems solved in the dissertation.

2.1 Background

In this section, we first define the key terms used throughout the dissertation, and then state the assumptions made. We conclude the section with a brief primer on the inter-datacenter wide area networks of the commercial cloud providers. With this we aim to strengthen the fundamentals in the area of research so that the essence of the literature survey in Section 2.2, can truly be captured.

2.1.1 Definitions

The definitions are divided into five sections. In the first section, we define the fundamental terms used in cloud computing. In the second section, we define the most commonly used objectives in a traffic engineering system. In the third section, we define the type of workloads seen in a cloud computing environment. In the fourth section, we define the type of deadlines a workload may have. Lastly, in the fifth section, we define the type of system environments used in cloud computing. The key terms used throughout this dissertation are as follows:
Cloud Computing Fundamentals

- **Datacenter** - A datacenter (DC) is a facility consisting of servers (physical machines), storage and network devices (for example, switches, routers, and cables), power distribution systems and cooling systems.

- **Cloud Computing** - Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of three service models; mainly, infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) [9].

- **Intra-Datacenter Network** - The switch fabric in a datacenter interconnecting servers, storage devices is called an intra-datacenter network.

- **Inter-Datacenter Network** - The network connecting the geo-distributed datacenters of a cloud provider is called an inter-datacenter network.

- **Cloud Provider** - The entity that provides services in a cloud computing environment is called the cloud provider.

- **Tenant or Customer** - The entity that is the recipient of the services provided by a cloud provider is called a tenant. The term “tenant” is used to denote that the recipient is occupying space in terms of some resource(s) in the provider’s datacenter.

- **Workload** - When a tenant runs an instance of a software, it is called tenant’s workload [30].

- **Software Defined Network** - Software Defined Networking (SDN) proposes decoupling of networking control plane and data plane to allow direct programmability of network control function. Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network [32].

Objectives of a Traffic Engineering System

A traffic engineering system can have one or many objectives. We define the most commonly used objectives below.

- **Guarantee deadlines** - Given a workload request by a tenant, guarantee the completion time of the workload meets the deadline.
- **Guarantee bandwidth** - Given a workload request by a tenant, guarantee specified bandwidth for the workload.

- **Minimize Transfer Completion Time** - Given a workload request by a tenant, minimize the completion time for the workload.

- **Maximize Minimum Fairness** - Given any workload, maximize the minimum fraction of demand transferred among same request types.

- **Maximize Network Utilization** - Given any workload, maximize the number of bytes being transferred through the network over a specified interval of time.

- **Minimize Energy Consumption** - Given a workload request by a tenant, minimize the units of watts consumed for data transfer over a specified interval of time.

- **Minimize Bandwidth Usage** - Given a workload request by a tenant, minimize the number of bytes of data transferred over the network over a specified interval of time. This is a special case for VM migration type of workloads.

- **Minimize Transmission Cost** - Given a workload request by a tenant, minimize the cost of transmitting a number of bytes of data over the network over a specified interval of time. This objective is same as maximizing revenue.

- **Maximize Throughput** - Given a workload request by a tenant, maximize the throughput i.e., overall number of bytes of data over the network in a specified interval of time.

- **Maximize Request Acceptance Rate** - Given a workload request by a tenant, maximize the number of requests accepted in a specified interval of time.

**Deadlines**

We define below the types of deadlines often associated with workloads in a cloud computing environment.

- **Hard Deadline** The strict deadline, which if surpassed the utility of the transfer drops to zero immediately.

- **Soft Deadline** The flexible deadline, which if surpassed the utility of the transfer drops to zero gradually.
Types of Workload

In this section, we define the most commonly seen workloads in a cloud computing environment.

- **Type 1 - Hard Deadline Transfers** - A workload that can sustain no delay in data transfer completion i.e., has fixed or hard deadlines. If the deadline is not met the data transferred is deemed useless. For example, real-time data transfers such as interactive video games.

- **Type 2 - Soft Deadline Transfers** - A workload that can sustain some delay in data transfer completion i.e., has soft deadlines. If the soft deadline is not met, the data can still be transferred without much loss in utility. Upon violation of soft deadline some penalty may be incurred by the cloud provider, for example, VM migration.

- **Type 3 - Soft and Hard Transfers** - A workload that can sustain some delay in data transfer completion i.e., has both soft and hard deadlines. If the soft deadline is not met, the data can still be transferred without much loss in utility until the hard deadline. Upon violation of soft deadline some penalty may be incurred by the cloud provider. However, upon violation of the hard deadline the data transfer is deemed useless, for example, video transfers.

- **Type 4 - No Deadline Transfers** - A workload that can sustain any amount of delay in transfer completion, but for which the volume of data to be transferred is known before hand, for example, redundancy backups.

- **Type 5 - No Deadline No Volume Transfers** - A workload that can sustain any amount of delay in transfer completion, but for which the volume of data to be transferred **is not** known before hand, for example, HTTP chunked transfer.

Types of System Environment

We define below the types of system environments commonly used in a cloud computing environment.

- **Public Cloud** - Public cloud is the cloud infrastructure provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider [9].

- **Private Cloud** - The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (for example, business units). It may be owned,
managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises [9].

- **Hybrid Cloud** - The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability. [9]

- **Federated-cloud** - Different cloud providers have an agreement to share their resources. The user of the cloud’s resources or services is not aware if the provided resources or services are consumed from one cloud or the other [27].

- **Multi-Cloud** - Cloud providers do not have any agreement to share their resources. The user of the cloud’s resources or services or a third party is aware of the different clouds and is responsible to deal with provisioning and/or managing the said resources or services.

### 2.1.2 Assumptions

We make the following assumptions throughout the dissertation.

- For all workload requests, a request may originate either at a server or the datacenter gateway. Similarly, the endpoint could either be the destination datacenter gateway or a recipient server in a datacenter. For those interested in provision of guarantees to workloads in intra-datacenter networks, there is an extensive body of work done in the area [5], [7], [11], [18], [22], [31]. Hence, we do not spend any time on that. We assume a datacenter to be a black-box for the rest of the dissertation.

- In this work, we assume a workload is comprised of multiple flows. We do not consider flowlet or co-flow level granularity for simplicity. This forms a natural extension to our research direction and we leave it to future work.

- In this work, we do not consider the presence of interactive transfers. Interactive transfers are the smallest subset of traffic by volume and are most sensitive to delay. This workload type can not be handled by the logically centralized SDN controller, as it will incur additional delay en-route to and from the controller. In much of the related work, as we will see in the literature survey, a portion of link bandwidth is reserved for such transfers. The exact amount to be reserved is dynamically predicted based on historical data. Given how extensively (see [53], [25]) this problem has been studied in the past, we do not spend any time on it.

- In our system model, a flow can be split and routed along multiple paths, resulting in packet level reordering. We assume techniques like MPTCP [19] can be used to resolve the problem.
2.1.3 Commercial Inter-DC WANs

In this section, we describe the inter datacenter networks used by two major cloud providers, Google and Microsoft. We describe the infrastructure and the traffic engineering policies used by these providers.

- **Google’s Inter-DC WAN (B4)** - B4 [26] is Google’s inter-datacenter WAN that accounts for over 90% of internal application traffic. It provides connectivity among datacenters for asynchronous data copies, index pushes for interactive serving systems and end user data replication for availability. They use a separate WAN for user-facing networking and exchanging traffic with other networking domains. The user-facing WAN is much more dense and supports highest levels of availability. For bandwidth sharing purposes they use max-min fair solution that maximizes bandwidth utilization as long as further gain in utilization is not achieved by penalizing fair share of applications.

- **Microsoft’s Inter-DC WAN (SWAN)** - In SWAN [25] the authors identify two key reasons for low average utilization of inter-datacenter WANs, one being the lack of coordination between the services using the network. For example, delay tolerant services can be served when the demand for other services is high. Second, the distributed resource allocation model used, for example MPLS TE wherein no entity has the global view of the network hence, resulting in selection of globally sub-optimal routes. The authors thus propose software defined WAN i.e., SWAN by coordinating the sending rates of services and centrally configuring the network data plane. They classify traffic into interactive, elastic or background transfers. When computing allocated rate for services, their goal is to maximize network utilization subject to service priorities and approximate max-min fairness among same-priority services. SWAN estimates an interactive service’s demand based on its average usage in the last five minutes.

2.2 Literature Survey

In this survey, we are interested in the guarantees provided in inter-datacenter WANs in the context of different cloud computing environments. Depending on the types of objective optimized, types of workload considered and control knobs used, we analyze the characteristics of the traffic engineering systems used in inter-datacenter WANs. We also aim to identify open areas of research. We present the survey of 20 most relevant papers to our area of research. The survey is divided into four parts.

- **The objective met and system environment** - The top level questions we answer by this classification are: ‘What are the different types of traffic engineering objectives studied
in the inter-datacenter WAN?’ and ‘In the context of which system environment have these objectives been studied?’ Since the overarching goal of this dissertation is to provide traffic engineering for workloads in inter-datacenter WANs, it is useful to identify the objectives that have already been studied, and the context of cloud computing environments in which they have been studied.

- **The type of workload considered to meet the objectives** - The top level question we answer by this classification is ‘What are type of workloads in terms of deadline requirements and demand size that have been considered for providing TE in inter-datacenter WANs?’. Any traffic engineering system must clearly specify the characteristics of the workload for which it assumes responsibility. A lucid classification of the literature on the basis of workload considered is hence, necessary.

- **The system environment used, SLA guaranteed, methodologies for admission control, scheduling, path selection, buffer allocation proposed.** - The top level questions we answer by this classification are ‘What are the control knobs used by the system?’, ‘Who can turn these control knobs?’ and ‘What are the performance guarantees provided by the system in terms of SLAs?’. With these three questions we aim to capture the crux of the system proposed by a work. We intend to identify the hats worn by the authors while proposing the system, applicability of the work in the real world in the form of an SLA and the control knobs that the work plays with. All of which are essential metrics to understand any real-world TE system.

- **The performance evaluation characteristics used** - The top level question we answer by this classification is ‘What are the performance metrics and performance evaluation methodologies used in a work?’. This gives us evidence of the depth of evaluation done by a work, and an understanding of the overall performance of the system.

### 2.2.1 Classification by Objective and System Environment

In this section, we classify the literature in terms of TE objectives considered and cloud computing environment used. In Sections 2.1.1 and 2.1.1, we define the objectives and system environments respectively. With this classification, the goal is to analyze why some TE objectives are more widely studied than the others. We also aim to identify the key differences between the works which solve the same TE objective.

Ji et. al. [55] maximize the network throughput of all multicast inter-datacenter transfers subject to deadline constraints. They guarantee deadlines to the transfers by accepting only those transfers which meet the deadlines. They propose to use multiple Steiner trees for each
multicast transfer. They formulate the problem as a linear program and propose a heuristic to solve the problem. The problem is solved in a single cloud computing environment.

Noormohammadpour et. al. [45] minimize the cost of transmission of unicast transfers in inter-datacenter WANs and increase the number of transfers meeting their deadlines. The cost function is proportional to bandwidth allocated in a timeslot. The authors also aim to meet two secondary objectives - to reduce the time taken by the solution in allocating requests, and increase network utilization. They formulate the problem as a linear program to meet the primary objectives and propose a heuristic ‘Rapid Close to Deadline’ to meet the secondary objectives. The problem is solved in a single cloud computing environment.

Wu et. al. [51] maximize the overall weight of all the jobs accepted. In the special case where weight is 1, the total number of accepted jobs are maximized. Urgent requests are given higher weights. The weight of request, in other words, models its utility i.e., the benefit of completing the request. The problem is formulated as a linear program, with deadline modeled as one of its constraints, which maximizes the aggregate utility gain due to timely transfer completions before the specified deadlines. The problem is solved in a single cloud computing environment.

Zhang et. al. [53] minimize the completion time for the requests in inter-datacenter WANs, and a request is accepted only if its completion time is less than or equal to its deadline. They approximate minimal completion time by assigning different weights to the edges in the network. The problem is modeled as a linear program and deadline is modeled as a constraint. The authors also aim to increase WAN utilization and boost the acceptance rate of the requests. They do so by proposing a complementary heuristic wherein they reschedule some of the already allocated requests to make space for requests for which no allocation was found by the LP. The problem is solved in a single cloud computing environment.

Noormohammadpour et. al. [44] minimize transmission cost in selecting path to route multicast transfers while guaranteeing hard deadlines. The secondary objectives of this work include avoiding packet reordering and improving network utilization, that are met by using path selection heuristics. Noormohammadpour et. al. [49] minimize overall bandwidth usage along the forwarding trees used to route multicast transfers in inter-datacenter WANs while guaranteeing deadlines and increasing acceptance rate of multicast transfers. The authors assume a multicast transfer meets its deadlines if the transfer is completed to all the destinations in time less than the deadline. The authors formulate the problem as mixed linear integer program, and propose a system that employs a Steiner tree selection method, an ALAP based scheduling policy and a heuristic to increase network utilization to solve the problem. Both the problems are solved in a single cloud computing environment.

Jin et. al. [42] propose a problem of finding the optimal network state as an online optimization problem. Given a stream of new transfers arriving in a system, at each timeslot, the authors try to compute a network state that optimizes the average transfer completion time or
the number of transfers that meet their deadlines. The authors leverage SDN to get direct control of network devices and simplify network management; and modern reconfigurable optical add-drop multiplexer (ROADM) devices that allow fast remote re-configurations (for example, provisioning an optical circuit in tens to hundreds of milliseconds). Most of the other works discussed in the survey assume a fixed network layer topology.

Jalaparti et. al. [41] propose a system Pretium that combines traffic engineering with dynamic pricing of links. The main objectives of the system are to maximize the social welfare i.e., overall value minus costs of all transfers, and maximize cloud provider profits in terms of sum of price paid by customers minus costs incurred while guaranteeing deadlines. The system proposes three modules, namely, request admission interface, schedule adjustment module and a price computer to meet its proposed objectives. They outline a set of changes that should be made to the system to guarantee bandwidth to their customers.

Wang et. al. [33] propose a system that provides end-to-end delay bound for interactive flows in inter-datacenter WAN environments, while promising fairness of bandwidth allocation among interactive, elastic and background transfers. They also aim to optimize path selection such that overall network utilization is increased. They formulate the problem as mixed integer problem and convert it into a linear program.

Xu et. al. [58] solve the problem of scaling up a virtual cluster across geo-distributed datacenters, with and without VM migration (without changing original VM placement). Their main objective is to minimize inter-datacenter bandwidth cost and migration cost (if VM migration used) while fulfilling the bandwidth guarantee for each VM pair in the virtual cluster. Unlike other works, this is not a typical traffic engineering problem in inter-datacenter environments. The problem is solved in the context of single cloud environment.

Li et. al. [43] propose a system that provides bandwidth guarantee to delay tolerant transfers in inter-datacenter WAN environment. Their primary objective is to minimize the network cost due to bandwidth allocation. Their secondary objective is to avoid potential overload at low cost links. The problem is solved in a single cloud environment. The problem is formulated as an LP and solved using primal-dual method.

Lai et. al. [56] consider inter-VM transfers in their measurement study to identify inter-datacenter WAN characteristics and to identify if it is useful for either a cloud provider or tenant to relay traffic. Since it does not solve a TE problem we do not provide any classification for it in Tables 2.1 and 2.2.

Hong et. al. [25] and Jain et. al. [26] present design and implementation of SDN based private inter-datacenter WAN environments for Microsoft and Google respectively. In both the works, maximizing network utilization was the main traffic engineering objective.

Kandula et. al. [29] propose a system Tempus that provides a max min fraction guarantee for soft deadline flows as a fairness objective and maximizes overall network utilization in inter-
datacenter WAN environments. The problem is formulated as an LP and solved using mixed packing and covering method.

Li et. al. [34] propose a new problem of inter-datacenter as a service, wherein major cloud providers and ISP sell their residual bandwidth to application providers who require bandwidth reservation. The problem has two aspects, price computation and bandwidth reservation. The former is modeled using Nash bargaining method and demand segmentation, while for the latter the authors propose two algorithms. The objective of the problem can be most closely modeled as bandwidth guarantee in our classification.

Yaw et. al. [37] provides a solution to minimize cost while delaying end to end latency in a proxy network in a multi-cloud environment. This is not designed as flow optimization problem, but rather deals with a special case for interactive transfers that have the option of using proxy servers in multi-cloud networks to reduce their end-to-end latency; the authors basically answer whether there is any utility in using multi-cloud networks for the same. The main objectives are cost minimization and overall latency reduction.

Chen et. al. [39] propose a system Karuna that provides a minimal impact congestion control protocol for rate-control for deadline flows in intra-DC networks, they also minimize flow completion times using a shortest job first based scheduling algorithm. Unlike the other problems studied in this section, this work focuses solely on traffic in datacenter networks, and is not aimed to provide scheduling, routing and admission control decisions for inter-datacenter flows.

Sun et. al. [50] solves the problem of virtual cluster placement, scale-up, scale-down, and virtual cluster consolidation in a federated-cloud computing environment. It does not solve any traffic engineering problem. The main objectives of this work are - overall operational cost reduction and reduction in energy consumption, which the authors achieve by employing VM migration. The problem is solved in a federated-cloud environment.

Luo et. al. [57] maximize the utility for each deadline transfer in inter-datacenter WAN environments, where the utility is defined as a difference of the request’s revenue and penalty. The authors do not spend much time in modeling revenue and penalty, but they specify it as a function of the budget value associated with a request which is vaguely defined as being dependent on traffic size, the priority class of the traffic, the precise latency requirements and/or the transfer deadlines, etc. Hence, we categorize the traffic into maximizing throughput, which in this case only refers to requests with urgent deadlines.

In this section, we broadly classified the literature on the basis of the primary objective achieved and the system environment considered. We identified nine objectives from the literature reviewed. Guarantees provided were limited to deadlines or bandwidth, except for Pretium [41], which primarily provided deadline guarantees, and specified additional extensions that must be done to provide bandwidth guarantees. There has been a lot of work in leveraging
Table 2.1: Literature Survey w.r.t the objective met and system environment considered. In every cell, S - Single cloud, M - Multi cloud, F - Federated-cloud.

<table>
<thead>
<tr>
<th>Papers</th>
<th>Guarantee Deadlines</th>
<th>Guarantee Bandwidth</th>
<th>Maximize WAN Utilization</th>
<th>Minimize Energy Consumption</th>
<th>Minimize Transfer Completion Time</th>
<th>Minimize Bandwidth Usage</th>
<th>Minimize Transmission Cost</th>
<th>Maximize Throughput</th>
<th>Maximize Request Acceptance Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ji et. al. [55]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et. al. [45]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wu et. al. [51]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhang et. al. [53]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et. al. [44]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et. al. [49]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et. al. [48]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jin et. al. [42]</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Jalaparti et. al. [41]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et. al. [33]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xu et. al. [58]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Li et. al. [43]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Jain et. al. [26]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hong et. al. [25]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kaundula et. al. [29]</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li et. al. [34]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yaw et. al. [37]</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chen et. al. [39]</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun et. al. [50]</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luo et. al. [57]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
</tr>
</tbody>
</table>

SDN to maximize WAN utilization either as a sole objective or in conjunction with alternate objectives. Most of the works use a variation of a greedy pull back scheme, wherein if the bandwidth in the current time slot is available, the request scheduled into the future is pulled back in the current time-slot to use the previously un-utilized capacity. Works like B4 [26] and SWAN [25] do not provide guarantees or plan for the future, they both use a variant of greedy scheme for request allocation by multipathing, such that the network utilization is maximized in the current timeslot.

As can be seen in Table 2.1, there has not been much work done in the context of traffic engineering objectives in federated and multi-cloud. Sun et. al. [50] tackle the problem of handling resource (compute and bandwidth) provisioning and scaling to minimize cost and energy consumption in a federated datacenter environment using VM migration. They do not deal with guarantees in federated environment. Yaw et. al. [37] solve the problem of minimizing latency and rental costs for interactive requests by relaying traffic over a multi-cloud network. However, for their calculation, their system model assumes no control knobs at the edge of the cloud, which makes sense, since the problem is solved from a tenant perspective.
2.2.2 Classification by Type of Workload

In this section, we classify the literature based on the type of workloads. Since we are primarily focused on deadline aware schemes, the broad categorization of workload is done on the basis of deadlines. We define the types of workload and the types of deadlines in Section 2.1.1.

Ji et. al. [55] maximize throughput and guarantee deadlines for multicast flows in inter-datacenter WANs. The authors focus on elastic and background transfers with deadlines. They do not distinguish between types of deadlines. For transfers which do not have any associated deadlines, the authors assume relatively large deadlines. The distinctive feature of this work is that it is specifically designed for guaranteeing deadlines for multicast transfers. The authors define meeting guarantees for each multicast transfer, when all destinations receive the overall data before a particular time.

Noormohammadpour et. al. [45] minimize cost of transmission of transfers and guarantee deadlines. However, the authors consider only elastic transfers with hard deadlines.

Wu et. al. [51] maximize the utility of all accepted requests for bulk transfers with hard deadlines. They consider fixed deadlines, however they categorize the traffic on the basis of tightness of deadlines. They do not consider any other type of workload.

Zhang et. al. [53] consider both hard and soft deadlines to minimize the completion time for the transfers. However, they only consider either hard deadline requests or soft deadline requests in their system, the two types of request are not assumed to coexist. The authors assume the presence of interactive transfers and background transfers. For the former they reserve bandwidth in the network, while the latter is served in a best-effort manner.

Noormohammadpour et. al. in both [49], [44] only consider requests with hard deadlines. While the former provides a complete system to allocate multicast requests in inter-datacenter WANs such that bandwidth usage is minimized, the latter only provides a routing solution that speeds up the allocation time and selects a minimum cost transmission path for unicast transfers. In fact, Noormohammadpour et. al. modify DCRoute [44] such that it can be used in DDCCast [49]. Noormohammadpour et. al. [48] consider deadline and non-deadline flows simultaneously to evaluate which scheduling policy works best for different request types by fixing the remaining control knobs of admission control and routing.

Jin et. al. [42] solve the problem of finding an optimal network layer or WAN topology such that transfer completion time is minimized for non-deadline transfers or acceptance rate is increased for deadline transfers. The authors do consider the presence of both hard deadline requests and no deadline requests, but they do not provide a solution for both simultaneously. Depending on the type of traffic, a corresponding scheduling algorithm is used with shortest path routing. Obtaining an optimal network state is the focus of the problem rather than providing guarantees for different transfer types.
Jalaparti et. al. [41] assume the presence of both hard deadline and soft deadline transfers, however the only guarantee provided is transfer of fraction before a deadline. If the request has a transfer size that fits into the upper bound specified by the system, the request is accepted and provided the guarantee in terms of an SLA contract.

Wang et. al. [33] consider three broad categories of traffic in inter-datacenter environment much like most of the contemporary works. They consider interactive, elastic and background transfers. While the first is categorized by tight hard deadlines, elastic transfers usually are associated with flexible deadlines while background transfers are usually large transfers which do not require any deadline. They consider all three requests present simultaneously and provide deadline guarantee for interactive transfer, while promising intra-class fairness for all transfers.

Xu et. al. [58] only consider a specific type of workload for solving the problem of scaling-up virtual clusters in an inter-datacenter environment i.e., VM migration. This type of workload is studied in isolation in this work and is provided with a bandwidth guarantee for each VM pair. Since this traffic type does not have any associated deadline, we categorize it as a type 4 request.

Li et. al. [43] only consider delay tolerant transfers or background traffic in their work. They provide bandwidth guarantee to such transfers while minimizing the overall network cost.

Jain et. al. [26] and Hong. et. al. [25] specify the presence of interactive, elastic and background transfers in their networks, but do not provide guarantees for any traffic type.

Kandula et. al. [29] propose a system Tempus that provides fractional guarantees for long running flows with soft deadlines in inter-datacenter environment. They also consider the presence of high priority flows, for which they reserve capacity by predicting future demand. They do not consider any other type of traffic and hence, do not provide any other guarantee.

Li et. al. [34] consider requests from application which require bandwidth reservation. These requests are met by using inter-datacenter as a service available from cloud providers and ISP. The authors consider only bandwidth requests which can be mostly modeled in our classification as no deadline requests with known volume.

Yaw et. al. [37] solve the problem of minimizing cost in selecting suitable proxies in a multi cloud environment for latency sensitive communication between a client and host. The authors assume interactive traffic, which can be most closely classified as hard deadline traffic.

Chen et. al. [39] is one of the few works that provides opportunities for trade-off between deadline and non-deadline flows. They consider requests with hard deadlines, no deadline requests with known volume and no deadline requests with unknown volume. They solve problems of congestion control for rate-control for deadline flows and flow scheduling to minimize flow completion times of non-deadline flows.

Luo et. al. [57] consider two requests types in their system, mainly hard deadline transfers and soft deadline transfers. They maximize the utility for each transfer. However, they do not
provide any guarantees for the distinct traffic types. They try to maximize the revenue due to
deadline met fraction of request and minimize the penalty due to the fraction of request which
did not meet its deadline, which means a request can not meet its hard deadline and still be accepted.

In this section, we further classify the work on the basis of the workload type considered by
the authors. We broadly classify the workload into 5 categories as described in Section 2.1.1. The
objective of the problem solved largely depends on the type of workload considered. Amoeba
[25], and for the most of the literature that followed it, some capacity of the network is reserved
for high priority transfers, which may vary from one time slot to the other. The high priority
transfers are not managed by the SDN based traffic engineering solution but they are accounted
in the network state in every time-slot used by the SDN based TE to admit, schedule and route
the bulk transfers. As can be seen in Table 2.2, only seven works in the literature surveyed,
take into consideration the presence of mixed flows while meeting their objectives. Only Karuna
[39] takes into consideration the trade-offs involved when providing TE to deadline and non-
deadline flows, but it primarily focuses on congestion control and scheduling in intra-datacenter
networks. None of these works provide guarantees in terms of deadlines and fairness to mix-
flows with or without deadlines, with known or unknown volume to be transmitted, which as
we will discuss later, is not a trivial problem.

2.2.3 Classification by Control Knobs

In this section, we classify the literature on the basis of the control knobs it uses. The standard
control knobs for a TE system are admission control, scheduling and routing. We classify the
literature on the basis of these knobs and also specify the additional knobs if used. We provide
context in the form of the identifying whether the problem was tackled from a cloud provider,
ISP or tenant perspective, and by identifying the SLA, if any, guaranteed by the work. Based
on this information, we can form a holistic view to analyze the methodology used by a work to
solve the problem they proposed.

Ji et. al. [55] wear the hat of the cloud provider while solving the problem of guaranteeing
deadlines to multicast transfers in inter-datacenter WANs. While they do not explicitly specify
an SLA, they generalize providing deadlines as a requirement for most cloud provider SLAs.
The authors employ Steiner trees to route multicast transfers. To find a set of feasible Steiner
trees for each request they use DFS. The authors formulate the problem as a linear program
with the objective of maximizing throughput, which they solve using a heuristic to obtain the
admission control, scheduling and routing decisions. The heuristic removes low priority requests
from the input when the problem becomes infeasible. Other control knobs that the authors play
with are tightness of deadlines and number of destinations.
Table 2.2: Literature classification w.r.t the type of workload considered to meet the objectives. See Section 2.1.1 for workload definitions.

<table>
<thead>
<tr>
<th>Papers</th>
<th>Guarantee Deadlines</th>
<th>Minimize Bandwidth</th>
<th>Minimize WAN Utilization</th>
<th>Minimize Energy Consumption</th>
<th>Minimize Transfer Completion Time</th>
<th>Minimize Bandwidth Usage</th>
<th>Minimize Transmission Cost</th>
<th>Maximize Throughput</th>
<th>Maximize Request Acceptance Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ji et al. [55]</td>
<td>1,4</td>
<td>1,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et al. [45]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wu et al. [51]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhang et al. [53]</td>
<td>1,2</td>
<td>1,2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et al. [44]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et al. [49]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et al. [48]</td>
<td>1,2,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jin et al. [42]</td>
<td>1,4</td>
<td>1,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jalaparti et al. [41]</td>
<td>1,2</td>
<td>1,2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et al. [33]</td>
<td>1</td>
<td>1,2,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xu et al. [58]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li et al. [43]</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jain et al. [26]</td>
<td>1</td>
<td>1,2,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hong et al. [25]</td>
<td>1,2,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kandula et al. [29]</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li et al. [34]</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yaw et al. [37]</td>
<td>1,4,5</td>
<td>1,4,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chen et al. [39]</td>
<td>1,4,5</td>
<td>1,4,5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun et al. [50]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luo et al. [57]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Noormohammadpour et al. [45] also wear the hat of the cloud provider while guaranteeing deadlines to transfers. They guarantee deadlines in the form of constraints of the LP proposed, but do not explicitly propose an SLA. While the LP minimizes the cost of the transmission of hard deadline transfers, they employ a heuristic by the name of ‘Rapid Close to Deadline (RCD)’ to increase network utilization and decrease the time taken to find an allocation. The authors run a new LP for each new request, and find an allocation from the current time to its deadline with the residual network capacity as input. If the LP is infeasible, the request is rejected. They pull traffic from the future to current timeslot, if and only if the request for which the current timeslot along a path was reserved is unavailable at its scheduled start time. The authors play with the control knobs of admission control, scheduling and routing with RCD.

Wu et al. [51] wear the cloud provider’s hat while they meet their primary objective of maximizing the utility of accepted requests. They do not explicitly provide an SLA, but they do guarantee transfer completion by the hard deadline of an accepted request. What distinguishes this work from the others is that they split a job in equal sized chunks and identify their overall admission control, scheduling and routing decisions. The authors propose three algorithms to adjust these control knobs targeting varying levels of optimality and scalability. The authors in their system design, introduce a buffer management unit that manages all received chunks at
the gateway server in an intermediate datacenter, and temporarily stores them until they must
be transmitted. This is done to conform with their store-and-forward paradigm. The authors
employ tightness of deadlines as an additional control knob.

Zhang et. al. [53] also wear the cloud provider’s hat when they aim to minimize the comple-
tion time of requests with deadlines in inter-datacenter WANs. Their secondary goals for higher
network utilization and higher acceptance rate are also motivated from the same perspective.
Like the other works the authors do not explicitly propose an SLA, but they do promise guar-
antees for all accepted requests with hard deadlines. The authors propose two heuristics to
optimally decide the values for the control knobs of admission control, scheduling and routing.
Since the underlying problem is formulated as an LP, if that is infeasible, the request may or
may not be rejected. The authors propose a scalability vs. optimality trade-off wherein the
heuristic to reschedule a rejected request may become expensive in time with scale, but provide
a more optimal solution. The authors do not employ a store and forward paradigm.

Noormohammadpour et. al. [49] solve the problem of guaranteeing deadlines while minimiz-
ing bandwidth usage of inter-datacenter WANs when allocating multicast transfers by combin-
ing the work done in DCRoute [44] and RCD [45]. Every time a request arrives, the authors first
select a minimum weight Steiner tree with sufficient available bandwidth. The request is ad-
mitted if the total available bandwidth in the selected Steiner tree is more than that requested.
The request is then allocated on the basis of ALAP scheduling policy, where the resources are
not reserved until absolutely necessary. To increase utilization, the authors adjust the schedules
when there is unused capacity. Upon beginning of every timeslot, they pull traffic from closest
timeslots in the future over each point to multipoint (P2MP) forwarding tree and send it in
current timeslot, if there is available capacity along all edges of such a P2MP forwarding tree.
The authors also study the effect of number of destinations as an additional control knob. The
authors survey only scheduling policies in [48] keeping admission control and routing knobs as
fixed. In DCRoute [44], authors propose routing heuristics for unicast transfers keeping admis-
sion control and scheduling knobs fixed. All the three works assume a cloud provider perspective
and guarantee deadlines without proposing a formal SLA for it.

Jin et. al. [42] solve the problem of finding an optimal network state both from an ISP
and cloud provider perspective. Unlike other works the main control knobs the authors use are
the optical circuit configuration that builds a network layer topology, along with the path and
sending rate computation for transfers. To meet the deadline guarantee for deadline transfers
or to minimize transfer completion times for non-deadline transfers, the authors employ earliest
deadline first (EDF) or shortest job first (SJF) respectively. For routing, the authors assume
multipath routing and prioritize shortest paths when making routing decisions. They do not
specify admission control, hence, there is no guarantee if a request whose deadline was not met
is accepted or not.
Jalaparti et. al. [41] determine how a request would be served when it arrives. The customer is quoted a price menu where the price is the price to route x bytes of data. This menu depends on the current state of the network and the request’s parameters. The system also provides the maximum capacity that it can route before the deadline. The customer then chooses how much data to transfer (if any). Pretium explicitly guarantees in an SLA that minimum (max capacity proposed, capacity requested by customer) will be completed in a specified time interval. Anything beyond that capacity is routed on best-effort basis. The authors wear the hats of both the cloud provider and customer in designing their system. From a customer standpoint they generate a schedule that maximizes the total value across all customers, minus provider costs. From a cloud provider standpoint they maximize cloud profit which is a sum of payments made by the customers minus costs incurred. Admission control is done on the basis of the explicit contract presented to the customer. If the contract is accepted, the request is also accepted. In terms of scheduling and routing, the problem is formulated as a non convex optimization problem that is converted to a linear program. By running the LP in every timeslot these control knobs are adjusted. Additionally, authors focus on price computation as the control knob to maintain dynamic temporal link prices.

Wang et. al. [33] propose a system that jointly allocates bandwidth to interactive, elastic and background transfers, while providing an end-to-end delay bound for interactive transfers. The problem is formulated as a mixed integer problem which uses branch-and-bound method with linear programming to solve the problem with an LP solver. The authors assume a cloud provider perspective and guarantee deadline for only interactive transfers but do not specify a formal SLA. Weighted Fair Queueing (WFQ) has been widely used in the past as a service discipline to provide end-to-end delay bounds, similarly token bucket has been used for admission control. The authors use both these control knobs. Path selection is done such that network utilization is maximized.

Xu et. al. [58] propose two methods for scaling virtual cluster in inter-datacenter environments; one uses VM migration and the other does not. The problem is proposed mainly from a cloud provider’s point of view and does not provide any explicit SLA, but a bandwidth guarantee is provided for VM-pairs in the cluster being scaled up. To meet the objectives of minimizing bandwidth usage, transmission cost and guaranteeing bandwidth to VM pairs in the cluster, the authors only admit requests for which bandwidth can be guaranteed. For scheduling scaling requests the authors employ two strategies - First in First Out (FIFO) and Shortest Cost First (SCF). While the first policy is well known, in the second policy, authors sort all requests that have not been served in an ascending order based on the cost each will incur. Then select a request with the lowest cost and scale up its corresponding virtual cluster with the proposed algorithms. The path for allocation is selected based on minimum cost (bandwidth and migration(if applicable)) between the source and destination DCs. The authors also employ an
additional knob of VM migration.

Li et. al. [43] solve the problem of routing and assigning transmission rate for each flow for delay tolerant flows in inter-datacenter environment. The authors propose a cost-minimizing bandwidth guaranteed allocation method for bandwidth allocation, and extend that bandwidth allocation method with weight factors for each inter-datacenter link to meet the secondary objective of avoiding overloading of lower cost links. The main control knobs of routing and assigning transmission rate are adjusted using these two methods which employ a linear program to minimize the network cost. The problem is solved from an ISP perspective, with no formal SLA provided, but a bandwidth guarantee is provided.

Lai et. al. [56] is a measurement study that measures TCP throughput and traffic latency for inter-VM communication along inter-datacenter WANs to identify whether traffic must be relayed or not. It does not fit into our classification based on the control knobs.

Kandula et. al. [29] provide a spatial and temporal allocation for soft deadline requests, with maximum network utilization in inter-datacenter WAN environments. The authors solve the problem from a cloud provider perspective, as they assume all requests have soft deadlines and their secondary objective is to maximize network utilization. They do not guarantee a formal SLA but they do provide a guarantee of minimum fraction of transfer completion before deadline. In terms of scheduling and routing, the authors obtain these decisions by solving their LP in an online manner with a mixed packing and covering approach. In terms of admission control, they reject a request when the LP does not obtain a feasible solution.

Li et. al. [34] solve the problem of price computation for inter-datacenter as a service provider and bandwidth reservation from a cloud provider and ISP perspective. The system does not provide any explicit bandwidth guarantees. The main control knobs used are price computation and bandwidth reservation. Since, the problem does not fall into the category of a typical TE scheduling and routing problem, these control knobs are not used. The authors model the price computation problem based on Nash bargaining games and demand segmentation method. The authors propose two bandwidth reservation algorithms, one based on weighted fair bandwidth reservation wherein each application provider or customer reserves bandwidth from each iDaaS provider based on its relevant weight, which is calculated based on the demand of the application provider compared to all application providers. The second algorithm follows a max-min fair manner in reserving each iDaaS provider’s bandwidth with the computed bandwidth price and amount of bandwidth that each iDaaS provider is willing to share.

Yaw et. al. [37] answer two main questions - can small-scale application providers deploy multi-cloud proxy networks in a cost effective manner? Second, how can small-scale application providers allocate cloud resources to meet latency constraints of cooperative group applications? The authors propose a method to manage cloud resources (servers and communication links) to minimize their rental cost subject to a group latency threshold. Their model considers link and
forwarding latency as part of total latency. The problem is solved from a tenant perspective, and it does not provide any guarantees. The main control knobs is a low cost path selection for which they propose greedy solutions.

Chen et. al. [39] provide a congestion control protocol for intra-DC networks that minimizes congestion to allow rate-control for deadline flows in intra-DC networks; they also propose a flow scheduling mechanism to minimize the flow completion time for non-deadline flows. The work is done from a tenant perspective, wherein all types of tenant requests are considered and from a provider perspective in terms of improving deadline miss rate for deadline flows. With the help of minimal impact congestion control protocol the authors minimally allocate bandwidth to deadline flows, while they use a scheduling policy based shortest job first for scheduling non-deadline flows to reduce the flow completion times. The authors do not use the admission control and routing knobs in this system, and while they do guarantee deadline for deadline transfers they do not promise a formal SLA.

Luo et. al. [57] propose a performance guaranteed allocation where they maximize the utility of traffic containing hard and soft deadlines. The problem is formulated as an LP and solved using a primal dual approach. The authors approach this problem from a cloud provider perspective as their main objective is revenue maximization due to deadline met transfers. They do not guarantee a formal SLA, nor there is a guarantee of deadline modeled in the objective or constraints of the LP. In terms of the control knobs, the scheduling and routing decision at each timeslot is determined by the LP, for admission control the system rejects all requests with negative utilities i.e., with penalties higher than deadlines, where utility of a request is the difference between its revenue generating ability and the associated penalty for fractions of deadline transfers not met.

In this section, we delve deeper into each work and identify the hat worn by authors i.e., the perspective with which the problem is solved. It can be a cloud provider, internet service provider and/or tenant. We identify any service level agreements, or for that matter, any metric that was guaranteed to the consumers of the service in the inter-datacenter WAN. Then, we discuss briefly the admission control policy, the scheduling and routing policies if employed in the solution. In some works storage space is provided to transfers along the path they are scheduled in case of congestion or presence of higher priority transfers. This space is usually employed in the datacenters along the path allocated to the transfer. We specify this in the column ‘Buffer Allocation’ in Table 2.3. We then summarize the control knobs employed by the authors. There are only two works that take into account the ISP perspective, and two which formulate their problem from a tenant perspective. Most of the works provide deadline guarantees to hard deadline transfers, but these works assume the remaining workload as background transfers and allocate them on a best-effort basis. As for any traffic engineering problem the most common control knobs used in the literature are admission control, scheduling and routing.
Table 2.3: Classification by the perspective used, SLA guaranteed, if admission control, scheduling, path selection, buffer allocation are used along with additional control knobs.

<table>
<thead>
<tr>
<th>Ref.</th>
<th>CP or ISP</th>
<th>SLA Guaranteed</th>
<th>Admission Control</th>
<th>Scheduling Policy</th>
<th>Path Selection</th>
<th>Buffer Allocation</th>
<th>Additional Control Knobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ji et. al. [55]</td>
<td>CP</td>
<td>Deadline guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Tightness of deadlines, number of destinations</td>
</tr>
<tr>
<td>Noormohammadpour et. al. [45]</td>
<td>CP</td>
<td>Deadline guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Wu et. al. [51]</td>
<td>CP</td>
<td>Deadline guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Tightness of deadlines</td>
</tr>
<tr>
<td>Zhang et. al. [53]</td>
<td>CP</td>
<td>Deadline guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Noormohammadpour et. al. [44]</td>
<td>CP</td>
<td>Deadline guarantee</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Noormohammadpour et. al. [49]</td>
<td>CP</td>
<td>Deadline guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Noormohammadpour et. al. [48]</td>
<td>CP</td>
<td>None</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Jin et. al. [42]</td>
<td>ISP, CP</td>
<td>None</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Optical circuit setup, path and sending rate computation</td>
</tr>
<tr>
<td>Jalaparti et. al. [41]</td>
<td>CP</td>
<td>Fraction of transfer guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
<td>Price Computation</td>
</tr>
<tr>
<td>Wang et. al. [33]</td>
<td>CP</td>
<td>End-to-end delay guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Xu et. al. [58]</td>
<td>CP</td>
<td>Bandwidth guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>VM migration</td>
</tr>
<tr>
<td>Li et. al. [43]</td>
<td>CP</td>
<td>Bandwidth guarantee</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Jain et. al. [26]</td>
<td>CP</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Hong et. al. [25]</td>
<td>CP</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Kandula et. al. [29]</td>
<td>CP</td>
<td>Fraction guarantee</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Li et. al. [34]</td>
<td>CP/ISP</td>
<td>None</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>None</td>
<td>Price computation, Bandwidth reservation</td>
</tr>
<tr>
<td>Yaw et. al. [37]</td>
<td>Tenant</td>
<td>None</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Chen et. al. [39]</td>
<td>CP, Tenant</td>
<td>Deadline guarantee</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Sun et. al. [50]</td>
<td>CP</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>VM Placement, VM Consolidation, VM Migration</td>
</tr>
<tr>
<td>Luo et. al. [57]</td>
<td>CP</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
</tbody>
</table>
2.2.4 Classification by Performance Evaluation Characteristics

In this section, we classify the literature on the basis of the performance evaluation characteristics. With this we aim to analyze the performance metrics and evaluation methodology used. The WAN topologies and traffic type considered help us to understand better the scope of the system proposed.

Ji et. al. [55] use both experiments on a test-bed on GCP cloud and a simulation to evaluate their solution. They evaluate their solution based on two performance metrics, mainly inter-datacenter throughput and percentage of requests that meet the deadline. Since, they do not specify an online algorithm to solve the dynamic problem, they do not measure these metrics against arrival rate. They compare their work with DDCCast [49] and Amoeba [53]. While a comparison with the former makes sense as both the works solve the problem for multicast transfers, a comparison with Amoeba which considers only unicast transfers was not required. Their solution performed best in terms of the metrics considered when transfer requests had tighter deadlines and higher number of destinations. The authors do not specify the topology they used for the simulation, but they did describe their test-bed in terms of availability zones in which they deployed their VMs in Google Cloud Platform. In terms of workloads for the test-bed experiments the authors transferred 300 MB files, with deadlines chosen from uniform distribution. They associate each request with a priority value, which is also randomly generated. Workload characteristics for the simulation are not specified.

Noormohammadpour et. al. [45] use a simulation for performance evaluation and percentage of failed elastic requests, average link utilization, and average allocation time are the performance metrics used. The authors generate synthetic requests with arrival rate as a Poisson distribution with mean ranging between 1 and 8, the deadline is generated as an exponential distribution with mean of 12 timeslots, and the demand is also generated as an Exponential distribution with mean of 0.286th fraction of link capacity. The authors run the simulation for 576 timeslots i.e., 2 days, and take an average of three runs to calculate results. However, this evaluation is done only for a single link case which is a special case considered by the authors in which two datacenters are connected with a single link. In this case the proposed algorithm performs comparatively with Amoeba [53] in terms of fraction of requests rejected and network utilization, however it performs much better in terms of average time taken for allocation.

Wu et. al. [51] evaluate their proposed heuristics to maximize the utility of all accepted transfers in inter-datacenter networks by setting up a test-bed where IBM BladeCenter HS23 cluster is used to emulate a geo-distributed inter-datacenter network with 10 datacenters. The authors synthetically generate requests with arrival rate randomly chosen between 0 and 10 for each timeslot, files randomly generated between sizes 100 MB and 100 GB and the destination for each transfer also selected randomly. The performance metrics used were aggregate weight of
jobs accepted, number of jobs accepted and time taken to compute an allocation. The authors also measured resource usage at the gateway servers at each datacenter that handled data transmissions and control signalling. The authors compare their work with Postcard [15] in terms of time taken, in which all their proposed algorithms are superior. They also compare their work with Netstitcher [8] in terms of throughput, wherein the proposed algorithms again perform better.

Zhang et. al. [53] evaluate their proposed solution with both a test-bed and a simulation. The main performance metrics used are acceptance rate, network utilization and throughput. For the test-bed the authors use 30 servers to emulate an inter-datacenter WAN with 6 DCs and use iperf to generate TCP flows. For the simulation the authors use WAN topologies of two production inter-datacenter WANs i.e., Google’s inter-datacenter WAN, G-Scale and Microsoft’s IDN. In terms of workload in the simulation, the authors generate synthetic workloads with request arrival time modeled as a Poisson process wherein each request contains 1 - 6 transfers, the time between a deadline and start time is modeled as an Exponential process with a mean of one hour, average transfer throughput is also modeled as an Exponential process with mean 20 Gbps. The authors compare their solution Amoeba with SWAN [25] and Netstitcher [8] in terms of network utilization, throughput and request acceptance rate wherein Amoeba is the predictable winner. This is because the other two works do no provide guarantees for deadline transfers and because of mismatch in objectives the comparison is not very useful. The authors also compute the time taken for obtaining allocations, and identify the utilization gain, throughput gain and network utilization when the two proposed heuristics are employed together. The authors also use two pricing functions to evaluate the revenue gain over fixed bandwidth guarantee schemes and study the impact of larger timeslots and tighter deadlines.

Noormohammadpour et. al. [44] use a simulation to solve the routing problem in inter-datacenter network for deadline transfers with allocation time and request rejection rate as the main performance metrics. The traffic is generated synthetically with request arrival rate modeled as a Poisson process and demand as well as length of the request modeled as exponential process. The authors assume Google’s inter-datacenter topology, and randomly generated networks as input for the simulation. They compare their work with Amoeba [53] which is a complete scheduling and routing solution to establish higher speedup in allocation with lower request rejection rate for deadline transfers.

Noormohammadpour et. al. [49] also use a simulation to solve the routing problem in inter-datacenter network for multicast deadline transfers with total bandwidth used and total traffic admitted as the main performance metrics. The traffic is generated synthetically with request arrival rate modeled as a Poisson process and demand as well as length of the request modeled as exponential process. The simulation is run for 500 timeslots and results averaged over 10 runs. The work which solves the problem for multicast transfers is compared with Amoeba [53]
which solves the same problem for unicast transfers and DCRoute [44] which only solves the routing problem for unicast transfers, hence, not an apple to apple comparison. The authors also study the effect of number of destinations and request arrival rate on the chosen metrics.

Noormohammadpour et. al. [48] use a simulation to evaluate the scheduling policies proposed to solve the scheduling problem with flow completion time as a performance metric for non-deadline flows and deadline miss rate and average lateness (i.e., how long subsequent to deadlines flows are completed, in case deadlines were missed) for deadline flows. The authors assume no admission control and a single routing path. The problem is evaluated in highly loaded and lightly loaded network scenarios, and with different flow size distributions, where the former is modeled as Poisson process and the latter as Exponential and Pareto process. According to their performance evaluation the SRPT performs best for heavy-tailed flow sizes if reducing tail times is not an objective. SRPT also performs well for light-tailed distributions except for average lateness (in addition to tail times). EDF on the other hand, does not perform well under the variety of loads and distributions. Its effectiveness is dependent upon ratio of regular to deadline traffic.

Jin et. al. [42] use along with a test-bed that emulates the Internet2 topology, a flow-based simulator to evaluate Owan in large scale topologies. They use traffic from an ISP network as well as an inter-datacenter network from an internet service company. For deadline transfers the authors use the percentage of transfers that meet deadlines and the amount of bytes that finish before deadlines as performance metrics. For non-deadline traffic, the authors use transfer completion time as the metric. To generate traffic in the test-bed the authors use both iperf and a custom multi-threaded traffic generator to send emulated traffic. They compare their work with five other works, namely, MaxFlow: an approach that uses linear programming to maximize the total throughput for each time slot, MaxMinFract: an approach that uses linear programming to maximize the minimal fraction that a transfer can be served at each time slot, SWAN [25]: that uses linear programming to maximize the throughput while achieving approximate max-min fairness for each time slot, Tempus [29]: that first maximizes the minimal fraction a transfer can be served across all time slots and then maximizes the total number of bytes that can be satisfied, and Amoeba [53]. The comparison with the last two works is done only for deadline transfers, and Owan performs better than the two in terms of percentage of transfers that meet the deadlines and percentage of bytes that meet the deadlines.

Jalaparti et. al. [41] use a simulation to evaluate their proposed system, Pretium, that provides a framework for dynamic pricing with traffic engineering in inter-datacenter WAN environment. The authors use four metrics to analyze their system’s performance, namely, the social welfare from carrying traffic i.e., total value minus costs, cloud provider profits, network utilization, and fraction of requests that finish. The problem is built as an LP and solved using a Gurobi solver. The authors simulate traffic on the basis of a month-long traffic trace from a
production inter-datacenter WAN with Microsoft. The network has 106 nodes and 226 edges; where each node is a datacenter or a site. The authors compare their work with the offline optimal solution for the same online problem, another offline scheduling scheme with no pricing that solves a single offline LP to maximize the sum of total bytes transferred by the network minus the cost incurred, a region based pricing oracle that mimics the pricing scheme used in practice and schedules transfers to maximize amount of bytes before the respective per-request deadlines while accounting for the 95th percentile network costs. They also compare their work similarly with a time of day pricing scheme that accounts for peak and off-peak periods, a spot pricing based oracle. When compared in terms of the four performance metrics, Pretium performs better than the other pricing schemes, mainly because some of the schemes were offline and not practical, while the others did not employ a traffic engineering module. This is the only work we know that combines an exhaustive dynamic pricing scheme with traffic engineering in inter-datacenter WAN environment.

Wang et. al. [33] provide a system that provides end-to-end delay bound for interactive transfers in the presence of elastic and background transfers. They evaluate their work with a simulation using Google’s inter-datacenter WAN as the network topology. The traffic is synthetically generated. For interactive and elastic, their demands are generated randomly and scaled separately by a factor (such that at least one link’s load is nearly 99% under the multi-commodity flow model. For Background traffic, demands are also randomly generated but have larger volumes than the rest. In terms of performance metrics the authors select network utilization in terms of the total throughput of applications, end-to-end delay bound for interactive transfers, fairness of rate allocation for flows in different classes, impact of the size of candidate path on the total throughput and computational time. The authors compare their work with SWAN [25], MMF [2] and their own proposed base solution without the deadline constraints. While their solution performs better w.r.t all the metrics proposed, it is important to note that in the contemporary works, the interactive transfers are satisfied by reserving a percentage of bandwidth along all future links. It is also not very practical for a class of traffic i.e., highly sensitive to delay to have to weight for allocation computation by a centralized SDN controller and suffer the time cost.

Xu et. al. [58] use a simulation to evaluate their algorithms, the performance metrics used are - bandwidth cost of inter-datacenter links, the acceptance rate of scaling requests, and the cost of VM migrations. The authors simulate a random inter-datacenter network with 30 datacenters with bandwidth of inter-datacenter links varying between 1 Gbps and 10 Gbps. For the price per unit bandwidth, the authors use numbers from Amazon EC2 data pricing. The work is compared with a similar work done in the context of intra-DC network [46]. The authors consider that there are 10 jobs. Each job specifies a virtual cluster. Initially, they deploy only one VM in each datacenter for each job. The authors randomly generate 200 scaling requests,
each request attempts to scale up the virtual cluster of one job with the same increase ratio. They handle these scaling requests in a first-in-first-out (FIFO) manner. The cost of migrating one VM across different datacenters is always set to 1.

Li et. al. [43] evaluate their solution based on a simulation with bandwidth guarantee, total cost, the performance on avoiding the potential overload, and convergence as the main performance metrics. They assume the topology of inter-datacenter network based on Equinix. The bandwidth capacity on each link is set to be uniformly random within the range [1, 10] Gbps, and the cost of each link is set randomly with values in ranging from 50 to 100 for intercontinental links and 1 to 50 for other links. The traffic is synthetically generated assuming VM-VM communication as inter-datacenter flows with a bandwidth demand of 500 Mbps. The work is compared with Faircloud [17] which focuses on link-level fairness when allocating bandwidth on a congested link between two VMs.

Lai et. al. [56] quantify the bandwidth, latency, and opportunities for improving WAN performance via relaying traffic via datacenters in inter-datacenter WANs. Their measurements span three cloud providers (Amazon, Microsoft, and Google) and aim to measure WAN bandwidth and latency of paths interconnecting VMs, including both intra-datacenter connections and inter-datacenter connections. The authors conducted measurements over 40 datacenter regions across three cloud providers’ datacenters and covered trans-oceanic connections. They use the widely adopted measurement tool iperf3 to measure the bandwidth between VMs and use both ICMP-based ping and TCP-based hping3 to measure the RTT between two VMs. They measure latency and per-TCP throughput to understand the characteristics of inter-datacenter WANs.

Kandula et. al. [29] evaluate their system Tempus that provides fractional guarantees to deadline flows and maximizes network utilization in inter-datacenter WAN environment using a simulation. They use minimum fraction guaranteed, percentage of requests completing before their deadline, total utility, actual fraction guaranteed before deadline, running time and memory usage as key performance metrics. They use the topology and traffic matrices from one of Microsoft’s production wide area networks and also evaluate the system on larger synthetic topologies. To simulate the synthetic requests they use information for their own production WAN’s network traces. They compare their works with their own corresponding offline optimal problem, greedy max flow: which at each time step, first routes interactive traffic, then, given all the unsatisfied demands, it uses an LP solver to maximize the total flow that can be served in the current time step, it does not plan into the future; and greedy max min fraction guaranteed: which is similar to greedy max flow except that the LP solver’s goal is to maximize the minimum fraction served for all pending requests. The greedy max flow performs similar to Tempus in most metrics except minimum fraction guaranteed at deadline and in terms of average promise.
Li et. al. [34] propose a problem of inter-datacenter bandwidth as a service, where application providers can reserve bandwidth from cloud providers. The problem is divided into two parts, price computation and bandwidth reservation. The authors evaluate the system using a simulation with 100 iDaaS providers. Each iDaaS provider hosts an amount of bandwidth, and sells them to the application providers according to its own bandwidth pricing strategy. The bandwidth capacity for each iDaaS provider is set to random values. Their experiments are conducted on Yahoo! network flow datasets collected from border routers every 15-minutes during one day, containing both traffic between Yahoo! servers and client, and traffic between different Yahoo! datacenters. The authors use reserved bandwidth on iDaaS providers, per unit bandwidth price of iDaaS providers, revenue of iDaaS providers, bandwidth reservation of application providers and payment of application providers as the main performance metrics for their solution.

Yaw et. al. [37] propose a greedy algorithm for meeting the QoS needs of cooperative group applications in multi-cloud deployments. The work is evaluated using simulation on networks configured with connectivity and pricing models of Amazon, Microsoft, and Rackspace cloud providers. The authors estimate the latency between datacenters as their great-circle (orthodrome) distance divided by half the speed of light to account for switching delay and coiled fiber in fiber optic conduits. Server and bandwidth costs are taken from the parameters of three top cloud service providers in the US. The authors simulate groups of clients utilizing two types of applications, a gaming application with a latency threshold of 80 ms, group size of 12, and fixed session duration of 10 minutes and a video chat application with a latency threshold of 200 ms, group size of 10, and an exponentially distributed duration. To calculate VM proxy capacity for these applications, the authors empirically evaluate the number of sessions that can be proxied by a VM on each datacenter without queuing request traffic. To calculate the traffic costs the authors measured the mean volume of flows in these applications using pcap captures. The performance metrics used are percent of cost savings using a multi-cloud deployment over a single cloud and average latency.

Chen et. al. [39] propose Karuna that provides a rate control mechanism for deadline flows such that it is able to provide flow scheduling to non-deadline flows that minimizes their flow completion times in intra-DC networks. They evaluate the system Karuna using a test-bed and simulation. Deadline miss rate and 95th percentile flow completion time for non-deadline flows are the main performance metrics used. They also study the effect of explicit congestion notification and number of priority queues in the switches in their test-bed. For the simulation the authors use a 144-server spine-and-leaf fabric with 4 core switches, 9 top of rack switches, and 16 servers per top of rack switch. They use 10 Gbps link for server to top of rack links, and 40 Gbps for top of rack uplinks. To simulate traffic the authors use well known web search workload and a data mining workload. In these workloads, more than half of the flows are less
than 100KB in size. The authors also simulate long flows with sizes uniformly distributed from 1KB to 10MB, which means that half of the flows are larger than 5MB. Flow arrival is modeled as a Poisson process and the source and destination for each flow are chosen uniformly at random. The authors compare their work with DCTCP [6], $D^2TCP$ [20], $D^3$ [12] which do not consider mix-flows and pFabric [23] which simply prioritizes deadline flows over non-deadline traffic.

Sun et. al. [50] solve the virtual cluster scaling problem in a federated datacenter environment. To evaluate their solution, they assume each datacenter has a Fat-Tree topology composed of 128 physical servers connected through 80 switches. Each physical server has 30 units of memory. The bandwidth capacity of physical link in datacenter is 10 units. Each active server consumes energy of about 0.6 kW. They do not specify the inter cloud connectivity parameters used in the evaluation. The number of VMs in each VDC request is randomly set from 3 to 9. The node resource demand of each VM is generated randomly from 1 to 8 units. The bandwidth requirement of each virtual link is set randomly from 1 to 4 units. Energy consumption of each VM is proportional to its memory size. The authors compare their three proposed algorithms in the evaluation with total embedding cost (i.e., inter-datacenter and intra-datacenter embedding cost) and energy consumption as performance metrics.

Luo et. al. [57] evaluate their proposed framework that maximizes overall utility due to hard and soft deadlines using a simulation. They use the performance metrics of acceptance rate, network utilization and throughput with changing arrival rate. The problem is tested in three WAN topologies namely G-Scale [26], IDN [25] and Equinix. The bandwidth capacity of each link is set to be 16 Gbps. In terms of workload, authors generate synthetic traffic with transfer arrival modeled as a Poisson process where the arrival rate per timeslot is between 1-6 transfers. A pair of datacenters are randomly selected as the source and the destination for each transfer request. The start time of the request is the request arrival time and the deadline follows an exponential distribution. The deadline type of each transfer is randomly set to be either hard or soft. Besides, for each transfer, the average transfer throughput over the lifetime follows an exponential distribution with a mean of 2 Gbps, and the total volume of the transfer is calculated accordingly. Time-varying unit revenue, penalty and budget are randomly generated for each transfer, except for hard deadlines which have high penalty associated with them, and requests which require high throughput immediately have higher revenue values associated with them in earlier timeslots. The authors compare their work with their corresponding offline optimal solution, Amoeba [53] and Tempus [29]. Since Amoeba [53] only guarantees hard deadline requests and Tempus [29] only provides a fractional guarantee for soft deadline requests, the comparison is not fair given the mismatch of the overall system objectives.

In this section, we describe the evaluation methodologies used in the literature to evaluate the authors’ proposed solutions. We identify the performance metrics used, evaluation methods
employed which can be experimental or real world deployment, mathematical or simulation. The topologies used for the evaluation are outlined, and the workload-transfer characteristics for the work are identified. The most commonly used topology for evaluation using simulation in the literature is Google’s inter-datacenter WAN topology with 12 sites and 19 inter-datacenter links. Most works rely on simulation for evaluation and use network utilization, throughput and request acceptance rate as performance metrics if meeting deadlines is their primary objective. Request arrival rate is modeled extensively as Poisson process, and the length of transfers as well request throughput are modeled as Exponential distribution. Unless the work is a measurement or experimental study, the traffic is generated synthetically.

2.2.5 Summary

In this section, we summarize the literature surveyed in terms of their contributions and shortcomings.

Ji et al. [55] design a routing and scheduling heuristic for multicast transfers across geo-distributed datacenters to maximize network throughput, and meeting as many deadline transfers as possible. The authors also aim to minimize overhead due to data splitting and reordering by decreasing the number of Steiner trees to route the requests. They assume the priority of a request as an input, and specify no basis on which a request’s priority may be decided. Moreover, in their heuristic, the authors drop low priority requests when the linear program becomes infeasible, which may lead to starvation of low priority requests. The problem solved is essentially static in the sense that the authors assume requests arrive in the beginning of a timeslot i.e., all requests are known beforehand, which makes the problem less realistic.

Noormohammadpour et al. [45] design and evaluate a scheduling algorithm for elastic data transfers in a single link connecting two datacenters and in WAN environment to maximize link utilization and guarantee tenant deadlines with high speedup in allocation times. While the authors discuss potential approaches that can be used to solve such a problem, they only solve the problem for the unrealistic base case of two datacenters connected by a single link and fail to prove the feasibility of this approach when extended to a network of datacenters.

Wu et al. [51] propose a bulk data transfer problem in an inter-datacenter network, which they then solve by scheduling and routing equal sized chunks over time using a store and forward paradigm to maximize the overall transfer utility i.e., overall weight of the jobs accepted. The problem is formulated as a linear program and of the three algorithms proposed only one heuristic is established as a polynomial time complexity solution. In this heuristic the authors postpone the less urgent jobs to make room for urgent transfers. However, they also do opportunistic shifts to the current timeslot in case there is available bandwidth. Requests for which no solution is available are rejected. In the system design the authors lay out the system
### Table 2.4: Classification by performance evaluation characteristics.

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Performance Metrics</th>
<th>Evaluation Method</th>
<th>WAN Topologies</th>
<th>Traffic Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ji et. al. [55]</td>
<td>Inter-datacenter throughput, Percentage of requests that meet deadlines</td>
<td>Test-bed, Simulation</td>
<td>6 VMs on 6 DCs in GCP in 6 different regions</td>
<td>Synthetic; file replication</td>
</tr>
<tr>
<td>Noormohammadpour et al. [46]</td>
<td>Percentage rejection rate, average link utilization, and average allocation time</td>
<td>Simulation</td>
<td>2 datacenters connected by one link</td>
<td>Synthetic workload</td>
</tr>
<tr>
<td>Wu et. al. [51]</td>
<td>Aggregate weight of accepted jobs, Total number of accepted jobs, Scheduling delay, Resource consumption in gateway servers</td>
<td>Test-bed</td>
<td>10 datacenters with links not specified</td>
<td>Synthetic workload; file generation</td>
</tr>
<tr>
<td>Zhang et. al. [53]</td>
<td>Network utilization, Request acceptance rate, Throughput</td>
<td>Test-bed, Simulation</td>
<td>GScale, IDN, iperf (test-bed), Synthetic trace (simulation)</td>
<td></td>
</tr>
<tr>
<td>Noormohammadpour et al. [44]</td>
<td>Allocation time, Request rejection rate</td>
<td>Simulation</td>
<td>GScale, Random (4 networks with 5 to 20 nodes)</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Noormohammadpour et al. [49]</td>
<td>Total bandwidth used, Total traffic admitted</td>
<td>Simulation</td>
<td>GScale</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Noormohammadpour et al. [48]</td>
<td>Flow completion time of regular flows, Deadline miss rate and Average lateness of deadline flows</td>
<td>Simulation</td>
<td>2 datacenters connected by one link</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Jin et. al. [42]</td>
<td>Transfer completion time, Deadline meet rate, Size of transferred finished before deadlines</td>
<td>Test-bed, Simulation</td>
<td>Test-bed (nine routers and ROADM in Internet2 topology)</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Jalaparti et. al. [41]</td>
<td>Social welfare, Provider profit, Network utilization, Fraction of requests completed</td>
<td>Simulation</td>
<td>A production network</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Wang et. al. [33]</td>
<td>Network utilization, End-to-end delay bound for interactive class, Fairness of rate allocation, Total throughput due to different candidate path size, Computational time</td>
<td>Simulation</td>
<td>GScale</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Xu et. al. [58]</td>
<td>Bandwidth cost, Migration cost, Acceptance rate</td>
<td>Simulation</td>
<td>Full mesh topology of 30 datacenters</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Li et. al. [43]</td>
<td>Bandwidth guarantee, total cost, Link over- load cost, and convergence</td>
<td>Mathematical, Simulation</td>
<td>Equinix</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Lai et. al. [50]</td>
<td>TCP throughput, Latency</td>
<td>Experimental</td>
<td>3 VMs on 40 DCs owned by 3 CPs (AWS, Azure, Google)</td>
<td>TCP traffic using iperf, ICMP based ping and TCP based hping3</td>
</tr>
<tr>
<td>Jain et. al. [28]</td>
<td>Percentage utilization on WAN links, Percentage throughput</td>
<td>Actual deployment on Google’s WAN</td>
<td>10 sites with 46 unidirectional edges</td>
<td>Production flow data</td>
</tr>
<tr>
<td>Hong et. al. [25]</td>
<td>Throughput, Network Utilization, Fairness</td>
<td>Test-bed, Simulation</td>
<td>Test-bed (5 DCs across three continents), Simulation - Random, GSCale</td>
<td>Synthetic, Microbenchmarks</td>
</tr>
<tr>
<td>Kandula et. al. [29]</td>
<td>min. fraction guaranteed, % of requests completed before deadline, total utility, actual fraction guaranteed before deadline, running time and memory usage</td>
<td>Simulation</td>
<td>Microsoft WAN, synthetic WANs</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Li et. al. [44]</td>
<td>Avg. Reserved bandwidth, Avg. per unit bandwidth price, Avg. revenue, No. of demands satisfied, Fairness index of bandwidth reservation, Avg. payment</td>
<td>Simulation</td>
<td>100 IaaS providers</td>
<td>Real traces i.e., Yahoo! network flow datasets</td>
</tr>
</tbody>
</table>
architecture with a control flow in considerable detail. The authors assume only bulk transfers with hard deadlines in this work.

Zhang et. al. [53] propose an abstraction to allow tenants to specify their demands i.e., deadlines and data volume and for providers to honor the SLAs. The authors employ a temporal-spatial allocation algorithms (adaptive scheduling and opportunistic rescheduling) for on-line admission control to minimize transfer completion times while guaranteeing deadlines for all accepted requests. The algorithms when used together lead to a more optimal solution but the cloud provider pays the cost in the form of additional time taken to find the optimal allocations. Hence, AS can be employed in scenarios where optimality can be traded off for time. The authors describe a pricing model to encourage tenants to provide their true requirements by penalizing the tenants when they are unable to do so. Any request which is unable to specify its deadline is categorized as background transfer with no guarantee or fairness provisions. Additionally, in terms of deadline workloads, authors only consider the presence of hard deadline transfers.

Noormohammadpour et. al. [44] identify a routing algorithm for Inter-Datacenter networks which guarantees that transfers complete before their deadlines and avoids packet reordering by scheduling all packets of a request on the same path. Requests initially are allocated using ALAP but later maybe rescheduled. The authors only consider fixed deadlines; There is no preemption: once a request is allocated, it cannot be rescheduled. They do not formulate the problem as an LP and entirely rely on heuristics to obtain optimal solutions.

Noormohammadpour et. al. [49] focus on guaranteeing deadlines for point to multipoint transfer in inter-datacenter network while minimizing bandwidth usage and improving acceptance rate. They determine request’s feasibility, if feasible they find the forwarding tree to meet the objectives; The authors extend their previous work: DCRoute [44] - routes and schedules point to point traffic; DCCast - schedules P2MP traffic with focus on decreasing tail times. This work only focuses on point to multipoint traffic with fixed deadlines.

Noormohammadpour et. al. [48] survey scheduling policies and provide an experimental comparison to identify which policy works best in scenarios with both deadline and non-deadline transfers. The authors consider an online scenario where flows may arrive anytime with no prior knowledge of their arrival. They compare different scheduling policies and identify which policy suits best for which situation. The authors consider different permutations and combinations of
scheduling policies to cater each type of traffic such as First Come First Serve (FCFS), Shortest Remaining Processing Time (SRPT), Fair Sharing, two combinations of EDF and FCFS, former for deadline traffic and latter for non-deadline traffic with one case strictly prioritizing deadline transfers over non-deadline transfers and vice versa, two combinations of EDF and SRPT, former for deadline traffic and latter for non-deadline traffic with one case strictly prioritizing deadline transfers over non-deadline transfers and vice versa. The authors assume no admission control and assume all traffic uses a single path i.e., complexity of the network is abstracted, and focus only on scheduling traffic.

As can be gathered, Noormohammadpour et. al. divided the TE problem based on the two control knobs of scheduling and routing, they solved for one while keeping the other fixed. Only in DDCCast [49] both the control knobs are considered together to solve a problem applicable to the real-world, but that problem focuses only on multicast transfers.

Jin et. al. [42] propose a traffic management system Owan that optimizes wide-area bulk transfers with centralized joint control of the optical and network layers. Owan dynamically changes the network-layer topology by re-configuring the optical devices. The authors develop algorithms based on simulated annealing to jointly optimize optical circuit setup, routing and rate allocation, and dynamically adapt them to traffic demand changes. What makes the work interesting is the problem of guaranteeing deadlines is tackled from an ISP perspective with optical layer control assuming software defined optical WANs.

Jalaparti et. al. [41] present a framework, Pretium that combines dynamic pricing with traffic engineering for inter-datacenter WANs. Pretium maintains an internal time varying price for each link in the network, and it provides an interface to its customers to choose a promise from a price quote while expressing their request in terms of deadline and volume. A subsequent schedule adjustment module runs periodically to re-optimize flow allocations to manage available capacity and usage costs. Pretium uses the information of historical demands and prices accepted by users to dynamically estimate future demands and update prices. This allows them to quote higher prices during periods of high demand, preferentially serving requests with higher value and shifting traffic with lower value to low utilization periods. This work is focused mainly on modeling link costs and dynamic pricing for a system that provides guarantees to its users.

Wang et. al. [33] propose a system that guarantees deadlines for interactive flows in inter-datacenter environment in the presence of elastic as well as background transfers. The authors propose a utility function with different weights to reflect priorities of traffic classes in allocating bandwidth. The authors propose a concave log utility function to enforce proportional fairness principle among applications within the same traffic class. The authors use piece-wise linear functions to approximate the log utility function and transform the bi-linear term in the end-to-end delay constraint into linear constraints with binary variables. While the authors assume the presence of three classes of transfers they only provide deadline guarantee for interactive
transfers.

Xu et. al. [58] solve the problem of scaling up a virtual cluster across geo-distributed datacenters, with the aim of reducing the bandwidth cost of inter-datacenter links as well as fulfilling the guaranteed bandwidth requirement for each VM-pair of the cluster. They formulate the problem of virtual cluster scaling as an optimization problem and present two scaling algorithms: one to scale up a virtual cluster with the original VM placement unchanged, and the other is to scale the cluster with VM migration enabled. This problem, while not a typical TE problem in an inter-datacenter environment is an example of real workload in an inter-datacenter network which can utilize a pre-existing TE scheme, like many specified in this survey. However, authors provide a solution tailor-made for only virtual-cluster scaling. Also, they do not provide a solution for scaling down requests, but add that as a future work.

Li et. al. [43] propose a bandwidth allocation model to minimize the network cost and avoid the potential traffic overload on low cost links in inter-datacenter networks. To solve the optimization problem the authors develop a distributed algorithm by blending the advantages of alternating direction method of multipliers (ADMM) and the auxiliary variable method. They decompose the optimization problem into small sub-problems to make the problem feasible in real world. It is important to note that the work only provides bandwidth guarantees for a single type of workload i.e., delay tolerant transfers, and does not consider any other traffic type.

Lai et. al. [56] question the usefulness of relaying traffic via multiple datacenters to improve WAN utilization and minimize latency. Based on their measurements across 40 datacenters belonging to Amazon EC2, Microsoft Azure, and Google Cloud Platform the authors show that inter-datacenter bandwidth is spatially and temporally homogenous between two VMs in different datacenters, but can be heterogenous at TCP flow level. While the overall conclusion drawn by the authors is that there is not much utility in relaying traffic via datacenters for the tenants aiming to reduce transfer latency and for cloud provider aiming to improve WAN utilization, the experiments conducted by the authors are not comprehensive, as they themselves state. Also, the experiments were conducted from a tenant point of view and observable bandwidth from a tenant’s perspective only occupies a small part of WAN capacity, so measuring variations in the underlying WAN is a challenging problem. However, the authors add that there may be an opportunity to improve WAN bandwidth by detours when data transfers on large-scale VMs are considered.

Laoutaris et. al. [8] propose a system called Netstitcher for stitching together un-utilized bandwidth across different datacenters, and using it to carry inter-datacenter bulk traffic for backup, replication, or data migration applications. The authors use a store and forward paradigm. They target multipath bulk transfers in mostly predictable environments, which reduces their problem to a maximum flow formulation. They address unpredictability by re-
computing the schedule sufficiently often and when component failure is detected. NetStitcher uses information about future bandwidth availability to move data between datacenters at low cost; it does not support deadlines for requests.

Hong et. al. [25] present a system SWAN to maximize network utilization of Microsoft’s inter-datacenter networks by using an SDN based central controller that schedules traffic and decides rate limiting. These decisions are re-configuring in MAPE (Monitor Analyze Plan and Execute) loop such that the network’s data plane matches the traffic demand. The authors leave a small amount of scratch capacity on links to apply updates in a congestion-free manner, without making any assumptions about the order and timing of updates at individual switches. To allow scalability in the face of limited forwarding table capacity, SWAN greedily selects a small set of entries that can best satisfy current demand. It updates this set without disrupting traffic by leveraging a small amount of scratch capacity in forwarding tables. This work by Microsoft is one of two works that established the domain of SDN based WAN, and propelled the research forward in this area, as can be seen by the other works.

Jain et. al. [26] present B4, a private WAN connecting Google’s geo-distributed datacenters. The authors employ an SDN architecture using OpenFlow to control switches built from merchant silicon. Like SWAN the main objective is to maximize network utilization. The system uses multipath routing by splitting its flows to balance capacity against application priority and demands. Both Jain et. al. [26] and Hong et. al. [25] focus on improving the utilization of the inter-datacenter WAN. The resource allocation policies offered therein can be classified as greedy per timestep. They do not compute long-term allocation schedules which are needed to provide deadline SLAs for long-running requests.

Kandula et. al. [29] propose a WAN transfer framework called Tempus which generates scheduling, routing and admission control decisions for long-running transfers in space and time. The system accommodates both high priority or interactive traffic and long-running requests. Tempus is formulated as an LP and solved in an online manner with a mixed packing and covering method. The authors only consider requests with flexible deadlines in this system, as they do not guarantee full completion of transfer before their deadlines, rather a fractional completion of transfer before their deadline using a max-min-fairness approach. They also maximize the overall utility of all the requests, wherein they maximize the overall network utilization by accommodating requests which were not allocated by max-min-fairness method.

Li et. al. [34] propose an inter-datacenter network as a service (iDaaS), where application providers, make bandwidth reservations for bandwidth guarantees from those Internet giants to support their wide area traffic. The authors propose a bandwidth trading market of multiple iDaaS providers and application providers, and focus on the bandwidth pricing problem. The authors approach the pricing problem from a trade-off standpoint between iDaaS providers and application providers. They model the interaction between iDaaS providers and application
providers as a Stackelberg game, and analyze the existence and uniqueness of the equilibrium. The authors also present a bandwidth price computation algorithm based on geometrical Nash bargaining solution and the demand segmentation method. Based on the computed price, the authors present two bandwidth reservation algorithms, where each iDaaS provider’s bandwidth are reserved in a weighted fair manner and a max-min fair manner, respectively. The authors focus on pricing bandwidth guarantees in this new service type, but there is scope of provision of more guarantees and corresponding pricing schemes with this service.

Given a set of datacenters and groups of application clients, Yaw et. al. [37] propose that well-connected datacenters can be rented as traffic proxies to reduce client latency. Rental costs must be minimized while meeting the application specific latency needs. The authors propose a Cooperative Group Provisioning problem and show it is NP-hard to approximate within a constant factor. The authors introduce a greedy approach. The authors conclude that multi-cloud deployments increase the likelihood of meeting group latency thresholds with minimal cost increase compared to single-cloud deployments. However, the system does not account for policies and control knobs at the cloud edge in the context of architecture of multi-cloud. Essentially the authors model the system as an inter-datacenter network of a single cloud.

Chen et. al. [39] focus on the problem of how to schedule a mix of flows in intra-datacenter networks. They propose a system, Karuna that resolves the tension between different types of flows with a joint design of rate-control protocol i.e., Minimal-impact Congestion control Protocol (MCP) that handles deadline flows with as little bandwidth as possible and priority-based flow scheduling that minimizes the flow completion time of non-deadline flows with known and unknown volumes with limited priorities in commodity switches. Karuna is a mix-flow scheduling system that balances the interests of deadline and non-deadline flows by trading off the average performance of deadline traffic to improve the average and tail performance of non-deadline traffic. Unlike most of the work surveyed in this chapter, it is not designed to be an optimal flow scheduling algorithm, and the problem is solved in the context of an intra-datacenter network.

Sun et. al. [50] assume a resource request submitted to datacenter can be abstracted as a virtual datacenter (VDC) request which consists of virtual machines (VMs) connected through virtual switches, routers and links with guaranteed bandwidth. The authors solve the problem of evolving VDC provisioning i.e., embedding newly added virtual resource requests to the underlying physical infrastructure, allocate physical resources for the new added virtual nodes or virtual links, and reducing the amount of resource allocation or releasing the occupied physical resources for the resource requirement reduced or expired virtual components. The authors formulate the problem as an integer linear program with the objective of minimizing the total cost and energy consumption, while embedding the dynamic/ evolving VDC to the underlying distributed multiple datacenters. The total cost includes the inter-domain cost and intra-domain
cost. (Cost is cost of embedding and VM migration cost). However, it is important to notice that the problem solved in the federated-cloud context is a VM placement problem and not a traffic engineering problem.

Luo et. al. [57] address the challenge of online deadline-aware bulk transfers in the geo-distributed datacenters, while taking into consideration the requests with hard and soft deadlines. The authors propose a performance-guaranteed allocation (PGA) framework to perform the admission control, scheduling and routing for transfer requests in an online fashion. The problem is formulated as an LP and solved using the Primal-Dual approach. This is one of the few works that tackles resource allocation in inter-datacenter environments with mix of two deadline types of traffic. The authors associate each deadline request with a revenue and a penalty function, the values of these functions vary with the type of deadline considered. They maximize the sum of utility of all transfers, where utility of a request is defined as the difference between its revenue and penalty. The authors associate utility only with deadline requests, all other requests without any deadlines are treated in best-effort manner as a background request. The system model has been defined as such wherein there is no exact guarantee associated with either hard or soft deadline requests in terms of the LP solved, also the only difference between the two categories of the request is their revenue value. The authors also do not spend any time in modelling the revenue and penalty functions.

**Major Takeaways**

Now that we answered the questions we promised in the above sections, we summarize the takeaways here:

- **Objectives**: The objectives met in the literature vary depending on the type of hat authors choose to wear, the type of workload and cloud computing environment they consider. There is a lot of scope in modeling utility of request allocation dependent on these factors. For example, providing QoS to deadline traffic traversing in a federated-cloud environment from an ISP perspective.

- **Workload**: The problem of traffic engineering has not been studied in the context of mix-flows i.e., deadline flows (type 1, 2, 3) and non-deadline flows (type 4, 5) in inter-datacenter WAN environments. While deadline guarantees have been provided in many recent works, the requests which cannot be guaranteed deadline or are unable to specify deadline or don’t have deadlines are allocated on a best-effort basis. For deadline traffic alone, no work has taken into consideration hard and soft deadline transfers together and provide each of them their required guarantees.

- **Control Knobs**: The standard knobs for providing traffic engineering to traffic travers-
ing inter-datacenter networks still remain the same. However, with the added power of
centralized control of the inter-datacenter dataplane with SDN, these knobs are being
turned keeping in mind optimization objectives.

- **Performance Evaluation**: The optimization problems being solved in the area of traffic
ingineering in inter-datacenter networks are ultimately formulated as linear programs and
solved using off the shelf LP solvers such as CPLEX or Gurobi. The traffic is generated
randomly and network topologies are selected based on the newest information provided
by major cloud providers or ISP, else a randomized topology is used. However, since the
area of research is relatively new, there is not much work that is available for comparison
if we insist on aligning a proposed system’s objectives with those of our peers.
Chapter 3

Vritti: Static Problem in Single Cloud

In this chapter, we propose Vritti, a system that provides guarantees for mix-flows in inter-datacenter wide area network environments (definition of mix-flows provided in Section 3.2.1). Vritti is a traffic engineering system that generates admission control, scheduling and routing decisions for providing guarantees to mix-flows. We begin this chapter by describing Vritti’s design goals, which form the foundation for the remainder of the chapter. We then define the system model in terms of WAN traffic, network, revenue and fairness. In Section 3.3 we state the static problem, formulate the problem mathematically and identify the control knobs for the problem. We then establish the mathematical and business novelty of Vritti. In Section 3.4 we evaluate the solution to the proposed static problem using simulation. We conclude the chapter by motivating the need for solving the dynamic problem which we formulate in Chapter 4.

3.1 Design Goals

In this section we describe the design goals of the proposed system - Vritti. The objective is to manifest in Vritti our overarching research philosophy (see Section 1.3) of adaptability with changing customer demands and cloud provider goals, as well as evolving cloud computing system models. To do so, we model the tension between cloud customer and cloud provider goals in Chapters 3 and 4, modelling adaptability with changing customer demands and provider goals. While in Chapter 5 we adapt the system in the federated-cloud and multi-cloud environments, hence modelling adaptability with evolving cloud computing system models.

Historically, the work in the literature in inter-datacenter WANs has been associated with maximizing revenue, utility, request acceptance rate, network utilization or minimizing flow completion times. For example, PGA [57] maximized revenue but without clearly modelling revenue, Tempus [29] maximized utility which was formulated as a function of network utilization, whereas Pretium [41] maximized overall social welfare i.e., the value associated with each
byte of request minus incurred provider costs. Amoeba [53] minimized the flow completion times of requests by allocating them along shorter paths while Wu et. al. [51] maximized the request acceptance rate. The objective of the problem depends on the design goals of the proposed system, which instead is a reflection of the overarching research philosophy of the authors. We approach the problem of resource allocation in inter-datacenter wide area networks from two different perspectives - that of a cloud provider and that of a cloud customer. From a cloud provider perspective, our aim is to make the system usable for the cloud provider i.e., the system should be able to generate revenue. From a cloud customer perspective, our aim is to take into account that customers have different demands, we must be able to propose a system that can accommodate all kinds of customer demands. It is important to reiterate that the design goals chosen are a culmination of our research philosophy (see Section 1.3) and our findings from the literature survey (see Section 2.2.5), and the design goals of the previous work in the area were different from what we propose based on the hats worn by the authors (provider, customer, ISP) and their research philosophy.

Our proposed system Vritti has three design goals as enumerated below. The first and the second goals are - maximizing cloud provider revenue and providing fairness among mix-flow types, respectively. The tension between these conflicting objectives is apparent. To provide fairness among different request types, a provider may have to allocate low revenue requests in lieu of high revenue requests. On the contrary, if a provider only allocates high revenue requests to maximize their revenue, they may end up starving low revenue requests. A key question then is: how can a cloud provider schedule and route a mix of flows with and without deadlines such that their goal, whether it be to maximize revenue or provide fairness, can be met? This is a challenging problem from a cloud provider’s perspective, as it entails providing tailor-made promises for different tenant requests. Intuitively, since we are dealing here with contrasting objectives, we may have to trade-off one to achieve the other. Hence, our third design goal is formulated to provide a control knob to allow effective trade-off between cloud provider revenue and fairness among mix-flows.

3.1.1 Maximizing Provider Revenue

The cloud provider earns revenue in an inter-datacenter wide area network in primarily two ways, first, by providing network capacity to meet tenant demands and charging a certain price for it, second, by utilizing the network effectively and accommodating more requests. In this dissertation, we focus mainly on the former method of generating revenue, as our primary goal is to provide guarantees for mix-flows. While the objective of maximizing provider revenue seems straight-forward, the challenge lies in modelling the revenue as a function of tenant demands and network utilization, mainly because cloud providers such as Google [68], Amazon [62], IBM
etc. do not divulge their revenue formulations. The problem becomes more challenging as we must determine how revenue calculation varies for different flow types. We define provider revenue in detail in Section 3.2.3.

### 3.1.2 Providing Fairness

In a network such as an inter-datacenter wide area network, that accommodates different types of requests, fairness can be defined in many ways. However, in this dissertation we choose to define fairness in an inter-datacenter wide area network environment in terms of the share of network capacity a type of flow gets over a period of time. Our objective is to provide fairness for soft deadline and non-deadline flows, as these flows are more likely to be treated as best-effort traffic otherwise. In this dissertation, we assume five types of flows, we also call them mix-flows, as defined in Section 3.2.1. With these five flow types we try to cover the entire spectrum of specification of tenant request in terms of volume and deadline. Fairness can also be provided on the basis of flow size, flow duration, type of tenant, and many other factors. While the possibilities are interesting, it is a broad research area in itself, considering just the number of knobs that we have at our disposal to play with and hence, out of scope for this dissertation. Similar to formulating an accurate revenue definition, it is challenging to accurately define fairness among flows because of the already high number of moving pieces in a dynamic system such as an inter-datacenter wide area network. We define fairness model in Section 3.2.4.

### 3.1.3 Providing a Control Knob for Revenue vs. Fairness Trade-off

To harmoniously meet the two design goals of maximizing revenue and providing fairness, we must be able to effectively provide a trade-off between the two conflicting objectives. To do so, we must be able to provide a control knob for revenue vs. fairness trade-off. We define the control knob in the revenue definition in Section 3.2.3. Like any other control knob, the challenge with providing this control knob is identifying an optimal range for tuning the knob. We discuss this in more detail in Chapter 4.

### 3.2 System Model

In this section we describe Vritti’s system model. We describe the system in terms of the traffic that it caters to, the underlying network in which it functions and the revenue it generates for the cloud provider. The exact problem solved by the system is defined in Section 3.3.
### 3.2.1 WAN Traffic Model

As seen in the literature survey in Section 2.2, in most of the recent work, traffic in inter-datacenter WAN has been roughly categorized into predominantly three categories, interactive, elastic and background transfers. The work that has been done towards developing deadline-aware systems for admission control, scheduling and routing in an inter-datacenter environment only provides guarantees to workloads with deadlines, and treats the rest of the traffic on a best-effort basis. To the best of our knowledge, non-deadline transfers have not been been considered in conjunction with deadline transfers in the literature in inter-datacenter wide area network environment. Moreover, there has been no work that considers a complete spectrum of workload in terms of deadlines and volume to be transferred i.e., with soft deadlines, hard deadlines and no deadlines, with known and unknown volumes.

Based on the exhaustive literature survey done in Section 2.2, we introduce a complete classification of workload based on their deadlines and volume requirements. We categorize the flows into primarily five types i.e., flows with hard deadlines, flows with soft deadlines, flows with both soft and hard deadlines, flows with no deadlines but known volume, flows with no deadlines but unknown complete volume. The intuition behind categorizing flows into five specific types is to allow more fine grained request classification in terms of request demands. We want to provide flexibility to tenants in expressing their demands in terms of deadlines and volumes which should reduce incorrect estimates being provided for requests. It is well known that every tenant wants to maximize the value for the cost paid to a TE system. For example, if we give a tenant, who has a request for which they cannot estimate a deadline, an option to either specify their request as a deadline request or receive best-effort treatment, the tenant may choose to express their request as a deadline request with an inaccurate estimate of a deadline, thereby incurring high cost and unnecessarily pressuring the TE system. These scenarios mostly occur because tenants would prefer some guarantee over no guarantee at all, and they are willing to pay more for that [41]. We want to discourage such scenarios by providing apriori promises to all the five categories of flows.

One can argue that even five flow types is a very coarse grained classification to accommodate all types of tenant demands, and we admit that it is possible. The most fine grained approach will be to dynamically decide the guarantee for a request based on its unique characteristics. However, we feel those kind of guarantees will be difficult to promise apriori, and even if they were promised apriori, the implementation can have a very high memory usage owing to just the number of unique combinations that can be made on the basis of request characteristics.

Below we describe the five flow types. We also specify the guarantees provided to each flow type to meet their requirements. We summarize the flow types, their characteristics and guarantees in Table 3.2.
**Type 1: Flows with hard deadlines**

These are flows with known volumes and hard deadlines. This implies that if the complete volume is not transferred by the hard deadline, the transfer is useless. This type of flow requires an all or nothing guarantee, similar to the guarantee provided by Zhang et. al. [53] and Wu et. al. [51] i.e., the request should be admitted if and only if complete transfer can be guaranteed by the deadline, else it should be rejected. Any guarantee less than complete transfer by the hard deadline to type 1 flows is useless for the customer as the transfer has no utility without complete transfer. Tenants can however resubmit the request with a looser deadline or with a soft deadline upon rejection. Traffic such as data transfer for real time data analysis and search index synchronization of search services are examples of type 1 flows.

**Type 2: Flows with soft deadlines**

These are flows with known volumes and soft deadlines. The utility of these transfers may decrease gradually after the soft deadline is surpassed, however the transfers do not become useless after the soft deadline. For these types of requests a completion of the fraction of the demand is guaranteed by the soft deadline, while the remainder of the demand is treated as type 4 flow without deadline. By providing a guarantee for fraction of the volume, we ensure that a part of the request will be completed by the soft deadline, and by the definition of type 2 flows we assume some loss in utility for the remainder of the transfer. We could have provided full transfer completion guarantee by soft deadline for these flows but that would have been an overkill, especially when done in the presence of more urgent hard deadline flows. In contrast, providing no transfer guarantee would have been equivalent to treating soft deadline flows as no deadline flows. Hence we take the middle road, and provide partial transfer completion guarantee. Workload types such as redundancy backups and VM migration are examples of type 2 flows.

**Type 3: Flows with hard and soft deadlines**

These are flows with known volumes and have both soft and hard deadlines. The utility of transfer of such flows may decrease with time after the soft deadline at a specified rate, becoming zero after the hard deadline is surpassed. These types of flows are guaranteed a fraction of transfer completion by the soft deadline and full transfer completion by the hard deadline. Since these flow types have requirements of both type 1 and type 2 flows, their guarantee is also a combination of guarantees of type 1 and type 2 flows. Traffic such as video transfer or data-set transfer, wherein partial data retrieval by deadline is useful, but full transfer completion is required by a certain deadline, are modeled as type 3 flows.
Type 4: Flows without deadlines but with known volumes

Some requests may not have any deadline associated with them but the transfer volume is known beforehand, these requests can be categorized as type 4 flows. Since these requests do not have any deadlines associated with them, guaranteeing them transfer completion by a certain time would put unwarranted pressure on the system. However, treating these requests on a best-effort basis is also unjustified, as it could lead to request starvation. Hence we choose to provide fairness to these flow types. We describe our fairness model in Section 3.2.4. Traffic such as redundancy backup transfers is an example of type 4 flows.

Type 5: Flows without deadlines and without known volumes

Some traffic types do not have any deadlines and may not be able to specify the complete volume for the transfer. Similar to type 4 flows, for these flow types as well we provide fairness. A number of applications may not be able to provide size/deadline information at the start of the transfer, for example, HTTP Chunked Transfer or Database access.

Transfer Request Specification

In this section we provide a template for the tenants to specify their request requirements. The goal is to allow tenants to specify the requirements explicitly for their transfer requests. While all previous works assume tenants specify exact deadlines and volume for their requests, in the proposed request template we allow higher flexibility for transfer request specification.

Depending on the specification by the tenant the transfer is classified into one of the five traffic types as defined in Section 3.2.1. The provider guarantee will depend on the traffic type the transfer is categorized into. Unlike the coarse grained traffic classification done in previous works, the tenants can specify transfer requests with soft deadlines, hard deadlines, soft and hard deadlines or no deadline, with known volume or unknown complete volume. The tenants specify each request in the form of six-tuple \(\{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_1,t,i}, k_{d_2,t,i}, V_{t,i}\}\), see Table 3.1 for definitions. Additionally, we assume an upper bound \(K\) on time during which the system is active.
Table 3.1: Tenant Transfer Request Model.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{t,i}$</td>
<td>Source datacenter for the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$D_{t,i}$</td>
<td>Destination datacenter for the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$k_{start,t,i}$</td>
<td>Start time, when the $i^{th}$ request generated by tenant $t$ arrives</td>
</tr>
<tr>
<td>$k_{d1,t,i}$</td>
<td>First deadline of the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$k_{d2,t,i}$</td>
<td>Second deadline of the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$V_{t,i}$</td>
<td>Total volume of data for transfer requested by the $i^{th}$ request generated by tenant $t$</td>
</tr>
</tbody>
</table>

Table 3.2: Traffic Model Summary.

<table>
<thead>
<tr>
<th>Flow Type</th>
<th>Specification</th>
<th>Guarantee(s)</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>${S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d1,t,i}, k_{d2,t,i}, V_{t,i}}$</td>
<td>$V_{t,i}$ transmission completion by hard deadline guaranteed</td>
<td>Search index synchronization</td>
</tr>
<tr>
<td>Type 2</td>
<td>${S_{t,i}, D_{t,i}, k_{s,t,i}, N/A, k_{d2,t,i}, V_{t,i}}$</td>
<td>$\alpha_{t,i} \cdot V_{t,i}$ transmission completion by soft deadline guaranteed (where $\alpha_{t,i} \in [0,1]$ is fraction guaranteed).</td>
<td>VM migration</td>
</tr>
<tr>
<td>Type 3</td>
<td>${S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d1,t,i}, k_{d2,t,i}, V_{t,i}, \alpha_{t,i}, N/A}$</td>
<td>$\alpha_{t,i} \cdot V_{t,i}$ transmission completion by soft deadline guaranteed, $\alpha_{t,i} \cdot V_{t,i}$ transmission completion by hard deadline guaranteed</td>
<td>Video transfer</td>
</tr>
<tr>
<td>Type 4</td>
<td>${S_{t,i}, D_{t,i}, k_{s,t,i}, N/A, N/A, V_{t,i}, N/A, N/A}$</td>
<td>Shortest Flow Completion Time (Fairness)</td>
<td>Redundancy backups</td>
</tr>
<tr>
<td>Type 5</td>
<td>${S_{t,i}, D_{t,i}, k_{s,t,i}, N/A, N/A, N/A, N/A}$</td>
<td>Average Bandwidth (Fairness)</td>
<td>Database access</td>
</tr>
</tbody>
</table>

3.2.2 Network Model

Similar to previous work in the area of providing guarantees in inter-datacenter networks, the inter-datacenter wide area network is modeled in a spatial temporal fashion, where time is a discrete entity and divided into fixed size timeslots. The space i.e., the inter-datacenter wide area networks is modeled as a directed graph $G = (V, E)$ where $V$ represents set of nodes or datacenters and $E$ denotes set of edges or overlay links connecting two datacenters or links connecting a datacenter to an ISP. Each edge $e$ has a capacity $c(e, k)$ in timeslot $k$. Typically, this capacity is in the range of hundreds of Gbps and the number of sites or datacenters varies from tens to hundreds per network. In Figure 1.3, we show the real inter-datacenter network of
Google’s private inter-datacenter WAN called G-Scale.

Figure 3.1 illustrates the system architecture for Vritti which contains a logically centralized SDN controller, the brain of the system. The controller maintains global knowledge of the underlying network including network utilization in every timeslot. It computes mainly three decisions, admission control, scheduling and routing for each incoming request. The network database stores the updated network state. With the logically centralized SDN controller, we make the system fault tolerant.

Every node or datacenter in the network is equipped with a site-broker, it is the entity responsible for synchronizing the network state with the controller, and enforcing the decisions made by the controller on the underlying network.

The SD-WAN was first implemented in the real inter-datacenter WANs by Google [26] and Microsoft [25]. The design of the system architecture of Vritti is heavily influenced by the architecture proposed and implemented by Google and Microsoft in their production inter-datacenter WANs. Mainly because the architecture used by Google and Microsoft is suitable for the problem we are trying to solve and meeting our design goals.

Similar to [25], [26], [29], [53], [57], we assume k-shortest paths pre-established between each pair of datacenters in the network. This assumption helps in reducing the problem size, making the problem more computationally feasible.

One or more paths from these k-shortest paths can be used to meet the demand by a request between a pair of datacenters. By using multipath routing we increase the number of candidate paths for the TE system to find a feasible solution. We are aware that using multipath routing would result in re-ordering at the receiver, however, we assume a mechanism similar to MPTCP [10] is used to deal with this. We believe that added complexity of using MPTCP is a small price to pay for higher provider revenue.
3.2.3 Revenue Model

The cloud provider earns revenue in an inter-datacenter wide area network environment by allocating network capacity to the incoming requests for a given time. So it makes sense, that the revenue be charged on the basis of the volume of request transferred and the time taken to transfer the request. The larger the volume of request to be transferred and/or tighter the deadlines, the request would require a larger chunk of network capacity. Similarly, higher traffic on a specific path would warrant a higher revenue to be accrued from requests traversing that path. We generalize these fundamental notions to formulate revenue for all request types.

For hard deadline, soft deadline and soft-hard deadline requests i.e., type 1, 2, and 3 requests, we provide a transfer completion guarantee for the fraction of the request transferred before the deadline. Hence, the revenue formulated for these requests is defined in equation 3.1. Revenue is directly proportional to the volume transferred, and the average path cost of all the candidate paths and inversely proportional to the time taken for transfer completion. Similarly, for soft deadline requests i.e., type 2 requests the volume of fraction remainder from that guaranteed is used to calculate the revenue as formulated in equation 3.2.

For non-deadline and known volume requests and non-deadline unknown volume requests i.e., type 4 and 5 requests, revenue is calculated on the basis of volume to be transferred. For type 5 requests we convert the rate request specified by the tenant into volume request. We assume the deadline for the request as the total run time of the system for the purpose of this dissertation. In a real system, the deadline for type 4 and 5 requests can be set as the minimum time by which the provider intends to meet all background requests. The revenue for these
request types is formulated in equation 3.3.

One may argue that all three revenue formulations are dependent on volume of the request transferred, time duration of the transfer and average path cost for the transfer. Though all revenue formulations are a function of volume, time and average path cost, it is important to note that the request characteristics vary significantly in terms of volume and time based on their flow type. Type 4 and 5 requests do not have deadlines hence their revenue is inversely proportional to the difference of system run time and request arrival time. Typically the system run time would be much larger than the deadline associated with type 1, 2, 3 requests, hence the revenue for non-deadline transfers would be much lesser as compared to deadline requests.

The current cloud providers do not provide a formulation for their revenue calculation. Even if they did, as far as we know no cloud provider and no work in literature provides revenue calculation for five flow types. Hence, the formulation was intuitively based on the volume of the data transferred by a TE system and the time taken for the system to do so. Since revenue should also be a function of network utilization, we use average path cost i.e., the averaged cost of all candidate paths associated with a pair of datacenters, to account for network state. In this dissertation, for simplicity, we assume static values for path cost, but in the real inter-datacenter networks the path cost would be a function of network utilization.

The notations used for revenue calculation may be overwhelming for the reader. However, one needs to only grasp the intuition behind the revenue calculation to delve deeper.

\[
\begin{align*}
\text{R}(x) &= r_{\text{guar}}(t, i) \cdot X_{k,\text{guar},t1,t3} + r_{\text{fair},1,3}(t, i) \cdot X_{k,t2,t3} + (r_{\text{fair},4,5}(t, i) \cdot X_{k,t4,t5}) \cdot wt_{\text{fair}} \\
\end{align*}
\]

\[
\begin{align*}
   r_{\text{guar},1,3}(t, i) &= \frac{\alpha_{t,i} \cdot V_{t,i} \cdot \text{avg\_path\_cost}_{t,i}}{k_{d1,t,i} - k_{s,t,i}} \quad (3.1) \\
   r_{\text{fair},2,3}(t, i) &= \frac{(1 - \alpha_{t,i}) \cdot V_{t,i} \cdot \text{avg\_path\_cost}_{t,i}}{k_{d2,t,i} - k_{d1,t,i}} \quad (3.2) \\
   r_{\text{fair},4,5}(t, i) &= \frac{V_{t,i}}{k_{d2,t,i} - k_{s,t,i}} \cdot \text{avg\_path\_cost}_{t,i} \quad (3.3) \\
\end{align*}
\]
where:

\[ X_{k,t1,t3} = \sum_{t,i \in \text{type 1,3 jobs}} x(t, i, p, k) \]

\[ X_{k,t2,t3} = \sum_{t,i \in \text{type 2,3 jobs}} x(t, i, p, k) \]

\[ X_{k,t4,t5} = \sum_{t,i \in \text{type 4,5 jobs}} x(t, i, p, k) \]

\[ X_{k,all} = X_{k,t1,t3} + X_{k,t2,t3} + X_{k,t4,t5} \]

### 3.2.4 Fairness Model

We provide fairness by allocating network capacity to non-hard deadline requests in lieu of hard deadline requests, whenever the hard deadline requests cannot be allocated.

Hard deadline requests, as we will see in the problem formulation below in Section 3.3, are modeled with all or nothing constraints i.e., for every hard deadline request, the transfer must be completed before their hard deadline. If all the hard deadline requests cannot be met by their hard deadlines, the problem becomes infeasible. While an all or nothing approach allows us to provide guarantees for hard deadlines, they are prohibitive for non-hard deadline requests. For example, if three requests arrive in a timeslot, two of which are hard deadline requests and one is non-deadline request, if the two hard deadline requests cannot be met before the deadline, we will not be able to meet the non-deadline request as the problem becomes infeasible since the constraints for the hard deadline requests are violated. To avoid the waste of network capacity when the problem becomes infeasible, and moreover, to provide fairness to non-deadline requests, we update the constraints of the problem dynamically when the problem becomes infeasible.

Our intention is to provide fairness for non-hard deadline requests while providing guarantees for hard deadline requests. We want to provide fairness, while still providing a knob in the hand of the cloud provider to control how much fair they would want the system to be. To do so, we cannot implement a static fairness solution. We describe the fairness solution when solving the more realistic dynamic problem in Section 4.3.2.

The main price we would pay for the fairness solution is additional time required to find an allocation. We describe the complete solution in more detail in Section 4.3.2.
Table 3.3: Notations and their definitions.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>Tenant or a customer that generates requests</td>
</tr>
<tr>
<td>$R(t, i)$</td>
<td>Set of $i$ requests generated by tenant $t$</td>
</tr>
<tr>
<td>$K$</td>
<td>Time for which the system is active</td>
</tr>
<tr>
<td>$k_{s,t,i}$</td>
<td>Start time, when the $i^{th}$ request generated by tenant $t$ arrives</td>
</tr>
<tr>
<td>$k_{d1,t,i}$</td>
<td>First deadline of the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$k_{d2,t,i}$</td>
<td>Second deadline of the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$\alpha_{t,i}$</td>
<td>Fraction guaranteed to the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$V_{t,i}$</td>
<td>Total volume of data for transfer requested by the $i^{th}$ request</td>
</tr>
<tr>
<td>$S_{t,i}$</td>
<td>Source datacenter for the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$D_{t,i}$</td>
<td>Destination datacenter for the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$P_{t,i}$</td>
<td>Set of candidate paths for the $i^{th}$ request generated by tenant $t$</td>
</tr>
<tr>
<td>$x(t, i, p, k)$</td>
<td>Network traffic amount by the $i^{th}$ request generated by tenant $t$ along path $p$ in timeslot $k$.</td>
</tr>
<tr>
<td>$\mathbf{x}$</td>
<td>Vector of $x$ values denoting the traffic distribution along all available tunnel paths $P_{t,i}$ in the time span $[k_{s,t,i}, K]$</td>
</tr>
<tr>
<td>$\mathbf{X}_q$</td>
<td>Vector of all $\mathbf{x}$ values in space $q$</td>
</tr>
<tr>
<td>$r_{\text{guar},t1,t3}(t, i)$</td>
<td>Revenue generated by $i^{th}$ request generated by tenant $t$ along path $p$ in timeslot $k$ for hard deadline requests.</td>
</tr>
<tr>
<td>$r_{\text{fair},t2,t3}(t, i)$</td>
<td>Revenue generated by $i^{th}$ request generated by tenant $t$ along path $p$ in timeslot $k$ for type 2 and type 3 requests.</td>
</tr>
<tr>
<td>$r_{\text{fair},t4,t5}(t, i)$</td>
<td>Revenue generated by $i^{th}$ request generated by tenant $t$ along path $p$ in timeslot $k$ for type 4 and type 5 requests.</td>
</tr>
<tr>
<td>$c(e,k)$</td>
<td>Capacity of edge $e$ at timeslot $k$</td>
</tr>
<tr>
<td>$w_{\text{fair}}$</td>
<td>Fairness weight for type 4 and type 5 requests.</td>
</tr>
<tr>
<td>$X_{k,t1,t3}$</td>
<td>All type 1 and 3 requests generated in timeslot $k$.</td>
</tr>
<tr>
<td>$X_{k,t2,t3}$</td>
<td>All type 2 and 3 requests generated in timeslot $k$.</td>
</tr>
<tr>
<td>$X_{k,t4,t5}$</td>
<td>All type 4 and 5 requests generated in timeslot $k$.</td>
</tr>
<tr>
<td>$X_{k,all}$</td>
<td>All requests generated in timeslot $k$.</td>
</tr>
</tbody>
</table>

3.3 Static Problem

In this section we formulate the static problem. We state the exact problem statement, its mathematical formulation as a linear program and the control knobs used. We end this section by establishing the mathematical and business novelty of Vritti as compared to the closest works in the literature.
3.3.1 Exact Problem Statement

Given an inter-datacenter network connecting datacenters in a multi tenant environment, also given transfer requests by each tenant with requests arriving in an offline manner, meaning the system is aware of all future requests at system start time.

The objective is to identify admission control, scheduling and routing decisions for five request types such that all guarantees are met and revenue is maximized.

Admission control, scheduling and routing are the control knobs that determine the flow allocation.

Vritti solves the static problem in a single cloud environment as follows: At system start time, the system becomes aware of all the future requests. Given an inter-datacenter wide area network, and all the requests known to the controller in advance, the controller calculates the spatial and temporal allocation for all requests by solving the mathematical formulation of the problem. Based on the allocation computed by the controller, it is decided whether a request is accepted or not depending on whether the corresponding guarantee is met or not. These decisions are then relayed to the site broker, which enforces the decisions on the underlying network.

3.3.2 Mathematical Problem Formulation

We formulate the baseline static problem mathematically as a linear program with two objective functions. The first objective function maximizes the minimal fraction guaranteed for all soft deadline requests, and the second objective function maximizes the overall revenue as defined in Section 3.2.3. With the first objective function, we meet the fairness objective for type 2 and 3 requests in a max-min fairness manner, and by maximizing revenue, we directly maximize the throughput allocated to all the requests. Note, the $w_{fair}$ knob in the revenue definition in Section 3.2.3, this control knob is introduced for the provider to allow for an optimal allocation trade-off between revenue generated by the deadline requests and revenue generated by the non-deadline requests. The default value for $w_{fair}$ is set to 1 allowing equal treatment for all requests from revenue calculation standpoint.

There are eight constraints associated with the linear program, six of which are demand constraints for the five types of requests. In equation 3.7 we specify the demand constraint for the type 1 requests which states that for each type 1 request, the sum of flow allocation along the paths selected and timeslots selected between the request start time and the request’s second deadline, must be equal to the demand by the request. Equation 3.8 states that for each type 2 and 3 request, the sum of flow allocation along the paths selected and timeslots selected
between the request start time and the request’s first deadline must be greater than or equal to the fraction of the demand guaranteed for the request. Note that the fraction \( \alpha_{t,i} \) is calculated by the first LP specified in equation 3.5. Equation 3.10 states that for each type 3 request, the sum of flow allocation along the paths selected and timeslots selected between the request start time and the request’s second deadline, must be equal to the total demand of the request.

Equation 3.9 states that for each type 2 request, the sum of flow allocation along the paths selected and timeslots selected between the request start time and the total run time, must be less than or equal to the total demand of the request.

Equations 3.11 and 3.12 state that for each type 4 and 5 request, the sum of flow allocation along the paths selected and timeslots selected between the request start time and the total run time, must be less than or equal to the total demand of the request.

Equation 3.13 states that the sum of the flow allocation for all the requests by all the tenants, along the paths selected and all timeslots between system start time and the total run time, must be less than or equal to the total capacity of the edge along the path at that time.

Equation 3.14 states that flow allocation along the paths and timeslots selected must be greater than or equal to zero for all the requests for all the tenants.

\[
\max \min \alpha_{t,i} \quad \forall t, i \in \text{type 2 and 3 requests} \tag{3.5}
\]

\[
\max R(x) \quad \forall x \in X_q \tag{3.6}
\]

s.t.
\[ \sum_{t,i \in \text{type} \ 1 \ \text{req.} \ k_d} \sum_{p \in P_{t,i}} x(t,i,p,k) = V_{t,i} \text{(Demand - Type1)} \] (3.7)
\[ \sum_{t,i \in \text{type} \ 2,3 \ \text{req.} \ k_d} \sum_{p \in P_{t,i}} x(t,i,p,k) \geq \alpha_{t,i} \cdot V_{t,i} \ \forall t, i \text{(Demand - Type2,3)} \] (3.8)
\[ \sum_{t,i \in \text{type} \ 2 \ \text{req.} \ k_d} \sum_{p \in P_{t,i}} x(t,i,p,k) \leq V_{t,i} \ \forall t, i \text{(Demand - Type2)} \] (3.9)
\[ \sum_{t,i \in \text{type} \ 3 \ \text{req.} \ k_d} \sum_{p \in P_{t,i}} x(t,i,p,k) = V_{t,i} \ \forall t, i \text{(Demand - Type3)} \] (3.10)
\[ \sum_{t,i \in \text{type} \ 4 \ \text{req.} \ k_d} \sum_{p \in P_{t,i}} x(t,i,p,k) \leq V_{t,i} \ \forall t, i \text{(Demand - Type4)} \] (3.11)
\[ \sum_{t,i \in \text{type} \ 5 \ \text{req.} \ k_d} \sum_{p \in P_{t,i}} x(t,i,p,k) \leq V_{t,i} \ \forall t, i \text{(Demand - Type5)} \] (3.12)
\[ \sum_{t \sum_{i \sum_{e \in P_{t,i}}} x(t,i,p,k) \leq c(e) \ \forall e, k \text{(Capacity)} \] (3.13)
\[ x(t,i,p,k) \geq 0 \ \forall t,i,p \in P_{t,i}, k \in [k_{s,t,i}, \min(k_{d,t,i}, K)] \] (3.14)

**Static Problem Size**

In Table 3.4 we estimate the number of constraints and number of variables in the static problem. For a typical system with hundreds of datacenters, thousands of edges, tens of thousands of requests i.e., handful requests per datacenter pair and hundreds of time units, we get an LP with millions of variables and constraints. Hence, solving the static problem every time a request arrives, for hundreds of timeslots, is computationally infeasible. Thus, we must find a more feasible approach to solve the dynamic problem, which we explore in Chapter 4.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Number of Constraints</th>
<th>Number of Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>(num. of type k jobs) * (demand constraints for that type k job) + (num. of edges) * (num. of timeslots) + (num. of jobs) * (num. of timeslots)</td>
<td>(num. of paths) * (num. of jobs) * (num. of timeslots)</td>
</tr>
</tbody>
</table>
3.3.3 Control Knobs

When a request arrives in Vritti, the logically centralized SDN controller must be able to compute the admission control, scheduling and routing decisions for the request. These decisions are generated by solving the linear program specified in Section 3.3. Since the overall objective of the system is to provide guarantees for mix-flows in inter-datacenter environments, admission control, scheduling and routing are the main knobs that come into play from the time the system becomes aware of a request’s arrival, until its completion.

- **Admission Control:** Vritti’s admission controller which runs in the SDN controller admits a submitted request if and only if its corresponding guarantee is met. See Table 3.2 for the guarantees corresponding to request types. This is a key control knob, every request that is accepted contributes to the revenue of the cloud provider, and the network state is updated accordingly along future timeslots to meet the request.

In the mathematical formulation of the problem in Section 3.3.2, the admission control knob is the value of \( \sum_{p \in P_{i,t}} \sum_{t=k_{s,t,i}}^{K} x(t, i, p, k) \). If the value of the summation of the variable for a request \( i \) of tenant \( t \) along all paths and along all timeslots starting from the arrival time of the request is equal to zero, then that request is rejected. Additionally, for hard deadline requests, if the value of summation is less than one at the hard deadline, the request is rejected.

- **Scheduling and Routing:** The scheduling and routing knobs also run in the SDN controller. Every time a request arrives in the system, the corresponding schedule and route for the request must be computed such that the objective of the linear program is optimized and all constraints are satisfied.

In the mathematical formulation of the problem in Section 3.3.2, the scheduling and routing knob is represented by \( x(t, i, p, k) \) i.e., the flow allocation for request \( i \) of tenant \( t \) along path \( p \) at time \( k \). Each \( x(t, i, p, k) \) variables for a request with value greater than zero, indicates the fraction of request allocated at the time \( t \) and path \( p \).

These decisions, once computed by the SDN controller, are enforced onto the underlying network by the allocation enforcement module in the site broker located at each datacenter or node in the network.

3.3.4 Mathematical and Business Novelty

In this section we compare three works in literature with our proposed work as they are the closest to our problem area. We do this to justify the mathematical and business novelty of our problem statement.
Tempus [29] proposed by Kandula et. al. considers only transfers with soft deadlines as their traffic model, and as a fairness objective, maximizes the minimal fraction for each request, and then to utilize the network completely, it maximizes utility. The problem is formulated as a linear program with constraints for capacity, demand, fairness and utility. The LP is solved using packing and covering solvers.

PGA [57] proposed by Luo et. al. maximizes revenue due to hard and soft deadline requests, however the revenue definition has not been provided in the work. This problem is also formulated as an LP with capacity, demand and budget constraints. The authors use the primal and dual approach to solve the LP.

Amoeba [53] proposed by Zhang et. al. provides guarantees for hard deadline requests and treats all other requests in best-effort fashion. This problem is also formulated as an LP with an objective of minimizing the flow completion time, with capacity and demand constraints. The authors propose two heuristics to solve the problem.

Pretium [41] proposed by Jalaparti et. al. formulates a non-convex problem to maximize social welfare in inter-datacenter networks. The problem is solved by converting to an LP with capacity and demand constraints.

In terms of traffic model Tempus [29] only considers soft deadline requests, Amoeba [53] and Pretium [41] consider only hard deadline requests whereas PGA [57] considers both hard and soft deadline requests. In contrast, our proposed system $Vritti$ encompasses a full spectrum of requests in terms of deadline and volume.

In terms of guarantees, Tempus [29] and Pretium [41] guarantee fraction of transfer completion before deadline, while Amoeba [53] provides full transfer completion before the deadline. PGA [57] does not provide any guarantee for deadline requests in the LP formulation. $Vritti$ in contrast, provides tailor-made guarantees for the five types of requests considered.

In terms of fairness, only Tempus [29] and Pretium [41] have fairness objectives, wherein they provide fairness among soft deadline requests, whereas with $Vritti$, we model fairness among all types of requests considered.

In terms of system environment considered, as far as we know, $Vritti$ is the only system designed with the goal of being adaptable in federated and multi-cloud environments. We summarize the comparison of $Vritti$ with Tempus [29], PGA [57], Amoeba [53] and Pretium [41] in Tables 3.5, 3.6 and 3.7.
Table 3.5: Mathematical Model Differences with Tempus, PGA, Amoeba, Pretium with Vritti.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Tempus</th>
<th>PGA</th>
<th>Amoeba</th>
<th>Pretium</th>
<th>Vritti</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective Function</td>
<td>Maximize minimum fraction and maximize utility (Two LPs solved sequentially)</td>
<td>Maximize revenue</td>
<td>Minimize flow completion time</td>
<td>Maximize social welfare</td>
<td>Maximize revenue dependent on price per link, fairness, flow allocation</td>
</tr>
<tr>
<td>Constraints</td>
<td>Capacity, Demand, Fairness and Utility</td>
<td>Capacity, Demand, Budget</td>
<td>Capacity, Demand</td>
<td>Capacity, Demand</td>
<td>Capacity, Demand, Fairness</td>
</tr>
<tr>
<td>Problem Type</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>Non Convex</td>
<td>LP</td>
</tr>
<tr>
<td>Method to Solve</td>
<td>Packing and Covering Solvers</td>
<td>Primal and Dual problems solved</td>
<td>Heuristics proposed</td>
<td>Non Convex Problem Converted to LP</td>
<td>Heuristics</td>
</tr>
</tbody>
</table>

Table 3.6: Business Model Differences with Tempus, PGA, Amoeba, Pretium with Vritti.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Tempus</th>
<th>PGA</th>
<th>Amoeba</th>
<th>Pretium</th>
<th>Vritti</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Model</td>
<td>Type 2 traffic</td>
<td>Type 1, 2 traffic</td>
<td>Type 1 traffic</td>
<td>Type 1 traffic</td>
<td>Type 1,2,3,4,5 traffic</td>
</tr>
<tr>
<td>Guarantees</td>
<td>Guarantee fraction of demand</td>
<td>No exact guarantee</td>
<td>Guarantee only hard deadlines</td>
<td>Guarantee largest volume transfer by deadline</td>
<td>Tailor-made guarantees for type 1,5 traffic</td>
</tr>
<tr>
<td>Revenue Model</td>
<td>Not modeled</td>
<td>Not modeled in detail</td>
<td>Not modeled</td>
<td>Social welfare model based on per link per byte pricing</td>
<td>Revenue model based on request demands, and path costs</td>
</tr>
<tr>
<td>Fairness</td>
<td>Max min fairness</td>
<td>Not an objective</td>
<td>Not an objective</td>
<td>Modeled as social welfare</td>
<td>Modeled as in Section 3.2.4</td>
</tr>
</tbody>
</table>

Table 3.7: Paper Comparison in Multiple Business Models.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Single Cloud</th>
<th>Multi-cloud</th>
<th>Federated-cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tempus</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amoeba</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PGA</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pretium</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vritti</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

3.4 Simulation and Evaluation

In this section we describe the simulation environment, the performance metrics used, the top level questions, and the simulation results.
3.4.1 Simulation Environment

We simulate one production inter-datacenter WAN - G-Scale, Google’s inter-datacenter WAN [54]. We solve the linear program using a well known LP solver, CPLEX [66], assuming each link has a uniform capacity of 160 Gbps, based on the assumptions made by the previous work in the area. We generate traffic synthetically, modelling the arrival process of requests as a Poisson distribution with mean ranging between 1 to 10 requests per timeslot. Time is discretized in timeslots, where each timeslot represents five minutes. The arrival time of the request is its start time. We randomly select a pair of datacenters as the source and destination for each request. The type of each request is set randomly to hard, soft, soft-hard, no deadline volume and no deadline no volume. For each request, the average transfer throughput over the lifetime is modeled as an exponential distribution with a mean of 20 Gbps. The total volume of the transfer can be calculated accordingly. One can argue that by calculating volume backwards with a given deadline and throughput, we are not exerting any pressure on the system. We are providing on an average a bound on the volume, such that it can be completed on time if allocated a fixed throughput from start time to deadline (or in case of non-deadline transfers, total system run time). However no request is guaranteed a fixed throughput from start time till its deadline, hence the inherent variability in planning for the future still remains. The hard values used for request generation and network characteristics are chosen such that they are consistent with the production inter-datacenter networks such as Google’s G-Scale [54] and Microsoft’s IDN [25]. To summarize -

- **Request arrival process**: Request arrival process is modeled as a Poisson process with parameter $\lambda \in [1, 10]$.

- **Deadline**: The minimum duration of the request i.e., $k_{d2,t,i} - k_{s,t,i}$ is modeled as an Exponential distribution with the mean of one hour.

- **Volume**: Average throughput is modeled as an Exponential distribution with the mean of 20Gbps. The total volume is calculated as $\text{throughput} \ast (k_{d2,t,i} - k_{s,t,i})$.

Performance Metrics

The simulation is run for 30 timeslots and the results are averaged over 30 runs. The performance metrics used to evaluate the system are chosen with the objectives of assessing allocation of all request types, assessing the network state and feasibility of the solution.

- **Percentage average request acceptance rate**: The percentage of average acceptance rate is used to assess how many hard deadline requests are accepted by the system of the total requests received. The system accepts only those requests for which hard deadlines
can be guaranteed. Hence, request acceptance rate is the apt metric to evaluate the allocation of type 1 and type 3 requests. The acceptance rate is calculated at end of the 30th timeslot, as the sum of all the type 1 and type 3 requests accepted by the SDN controller. The average acceptance rate is calculated by averaging the value of acceptance rate for 30 runs.

- **Percentage average fraction guaranteed:** This metric is only applicable to soft deadline requests i.e., type 2 and type 3 requests. Since *Vritti* guarantees a fraction of requested transfer completion before the soft deadline, the average fraction guaranteed is the apt metric to evaluate soft deadline requests. The average fraction guaranteed in a run is calculated at end of the 30th timeslot, as the average of the fraction guaranteed by the soft deadline of all the type 2 and type 3 requests. The overall average fraction guaranteed is calculated by averaging the value of average fraction guaranteed for 30 runs.

- **Percentage average fraction of type 4 requests allocated:** Average fraction of type 4 requests allocated denotes the total fraction of all type 4 requests allocated by the system run time, averaged over total number of type 4 requests. With this metric we aim to assess how much network capacity is allocated to type 4 requests over system run time. The average fraction allocated in a run is calculated at end of the 30th timeslot, as the average of the fraction allocated by the end of the 30th timeslot of all the type 4 requests. The overall average fraction allocated is calculated by averaging the value of average fraction allocated for 30 runs.

- **Percentage average fraction of type 5 requests allocated:** Average fraction of type 5 requests allocated denotes the total fraction of all type 5 requests allocated by the system run time, averaged over total number of type 5 requests. With this metric we aim to assess how much network capacity is allocated to type 5 requests over system run time. The average fraction allocated in a run is calculated at end of the 30th timeslot, as the average of the fraction allocated by the end of the 30th timeslot of all the type 5 requests. The overall average fraction allocated is calculated by averaging the value of average fraction allocated for 30 runs.

- **Percentage average network utilization:** Network utilization at each timeslot is calculated as the average link utilization over all links in the network. Average network utilization is network utilization averaged over all timeslots. This value is averaged over 30 runs. Average network utilization reflects how lightly or heavily loaded the network was during the specific run. It is an important metric, as high network utilization may mean that the provider is obtaining a high return on investment for their network resource. Also, average network utilization has high inter-dependency with the aforementioned per-
formance metrics, and therefore is a crucial metric to understand the performance of an algorithm.

- **Average throughput per request (Gb/timeslot):** The average throughput per request is calculated at the end of 30th timeslot by averaging throughput allocated over all timeslots from the request start time until system run time for all requests. This value is divided by total number of requests in a run and then averaged over 30 runs. Average throughput represents overall throughput per request that was allocated in a run. This metric is interdependent on the average network utilization and the metrics for types 1 - 5 requests.

- **Average revenue:** Average revenue is calculated by averaging revenue generated in a timeslot by a request, averaged over all requests and over all timeslots.

- **Percentage time taken:** Time taken for a run is defined as the time taken by the controller to generate a solution. Time taken for all runs is calculated by averaging the time taken for 30 runs. It is used to assess the computational feasibility of an algorithm.

- **Percentage memory usage:** Memory usage is the percentage of the memory capacity used for the computation. This value is averaged over 30 runs. It is used to assess the computational feasibility of an algorithm.

### 3.4.2 Top Level Questions

In this section we pose the following specific questions. We answer these questions through extensive evaluation. We mainly aim to analyze the performance metrics with the request arrival rate. The other parameters that can be used for performance evaluation can be network characteristics such as network capacity, network topology, or request characteristics such as request diversity ratio, request volume, request deadlines and so on. However, we choose to only use request arrival rate as the parameter because, practically, the arrival rate can update dynamically without many constraints, unlike say, the network capacity, changing which is not easy for a cloud provider. Performance evaluation with the remaining metrics is deferred to Chapter 4 when we evaluate the more realistic dynamic problem.

1. How effective is the algorithm in providing guarantees to mix-flows?
   - Type 1 and 3 requests: How does request acceptance rate vary with request arrival rate?
   - Type 2 and 3 requests: How does average fraction guaranteed vary with request arrival rate?
2. How does network utilization vary with request arrival rate?

3. How does throughput vary with request arrival rate?

4. How does revenue vary with request arrival rate?

5. How does space and time overhead vary with request arrival rate?

6. How does the solution compare with current inter-DC TE practices?

3.4.3 Simulation Results

Q1. How effective is the algorithm in providing guarantees to mix-flows?

In this section we evaluate the performance of the solution to the static problem w.r.t guarantees provided to mix-flows.

We first evaluate the effectiveness of the solution for type 1 and type 3 flows i.e., hard deadline flows in terms of request acceptance rate. These requests are accepted if and only if their hard deadlines can be met. As can be seen in Figure 3.2, the acceptance rate of hard deadline requests decreases with increase in request arrival rate per timeslot. In terms of average fraction guaranteed to soft deadline requests i.e., type 2 and type 3 flows, as can be seen in Figure 3.3, the average fraction guaranteed decreases with request arrival rate per timeslot. A similar behaviour is seen for type 4 and type 5 requests i.e., as seen in Figures 3.4 and 3.5, the average fraction of request allocated for type 4 and type 5 requests decreases with increasing request arrival rate per timeslot. Intuitively, all the graphs should monotonically decrease with arrival rate, because as the number of requests arriving per timeslot increases, the pressure on the LP solver also increases in terms of problem size. However, we must take into account the randomness of the request characteristics that affect these graphs, the request diversity ratio, the source and destination for each request, the request volume, the deadlines are all randomly generated for each run. Most likely, it is due to this randomness we see the bumps and flatness in the graphs.
Figure 3.2:  Acceptance Rate for Type 1 and Type 3 Flows in G-Scale.

Figure 3.3:  Average Fraction Guaranteed for Type 2 and Type 3 Flows in G-Scale.
Figure 3.4: Average Fraction Allocated for Type 4 Flows in G-Scale.

Figure 3.5: Average Fraction Allocated for Type 5 Flows in G-Scale.
Q2. How does network utilization vary with request arrival rate?

Network utilization, as seen in Figure 3.6, increases with request arrival rate. Intuitively, average network utilization should increase with request arrival rate. However, as in the case of the previous graphs, we attribute the non-monotonic increasing behavior to the randomness of the request characteristics.

![Network Utilization](image)

**Figure 3.6:** Network Utilization in G-Scale.

Q3. How does throughput per request vary with request arrival rate?

The average throughput per request decreases with request acceptance rate. Intuitively, average throughput should decrease with request arrival rate. However, as in the case of the previous graphs, we attribute the non-monotonic decreasing behavior to the randomness of the request characteristics.
Q4. How does revenue vary with request arrival rate?

Revenue increases with request arrival rate. Intuitively, with higher request arrival rate, the load on the network increases and more requests are allocated, thus an increase in revenue with request arrival rate is expected. However, as in the case of the previous graphs, we attribute the non monotonic increasing behaviour to the randomness of the request characteristics.
Q5. How does space and time overhead vary with request arrival rate?

In this section we evaluate the time taken by the algorithm, along with CPU usage and memory usage. We run the solver on a Linux system with Intel Xeon Gold 6130 processor, 16 cores and 2.10 Ghz base processor frequency. The time taken for finding allocation for all requests in the static problem increases almost linearly with request arrival rate. This is because the problem size also increases in direct proportion to the request arrival rate, and larger the problem size, the more time the LP solver takes to find a solution. The CPU usage remains at 100% for all request arrival rates, while memory usage increases linearly with request arrival rate.

Figure 3.8: Revenue due to all Requests in G-Scale.
Figure 3.9: Time Taken for Request Allocation in G-Scale.

Figure 3.10: CPU Usage for Request Allocation in G-Scale.
Q6. How does the solution compare with current inter-DC TE practices?

The problem proposed in this dissertation provides guarantees for mix-flows in inter-datacenter networks with the objective of maximizing provider revenue and providing fairness among flow types. While there has been work done to provide deadline guarantees for hard deadline requests, and fairness to only soft deadline requests, no work has provided guarantees to mix-flows considering both deadline and non-deadline traffic. Since the work closest to our area of research does not align in terms of objectives used and traffic model considered, there is no possibility of apples to apples comparison. Hence, we do not spend any time in comparing our work with Amoeba [53], Tempus [29], Pretium [41] or PGA [57].

3.5 Conclusion

In this chapter we introduced the static problem solved by Vritti. The objective is to provide guarantees for mix-flows by maximizing revenue due to all traffic types and providing fairness among the five flow types. We defer dynamically meeting the fairness objective in more detail for the dynamic problem introduced in Chapter 4. We simulate the problem as an LP in inter-datacenter network G-Scale [26]. We solve the LP using standard LP solver, CPLEX for 30 timeslots, and average the results over 60 runs. As expected, the performance of the solution to
the static problem decreases with increasing arrival rate. This is because of the all or nothing guarantee provided to hard deadline requests. If the LP solver is unable to find a feasible allocation for a hard deadline requests, the LP becomes infeasible and no other request types are allocated. However, this behaviour is specific to only the static problem and can be remedied in the dynamic problem as we will see in the upcoming chapter.

We also see, as the number of requests arriving per timeslot increases, the problem becomes larger for the LP solver. As described in Section 3.3.2, it is infeasible to solve the static problem in the realistic scenario where future requests are not known to the system in advance. Hence, we must develop a solution to optimally solve the dynamic problem which we explore in Chapter 4.
Chapter 4

Vritti: Dynamic Problem in Single Cloud

In Chapter 3, we formulated the static problem as a linear program in a single cloud environment, and solved it using a well-known LP solver. When defining the static problem, we made an assumption that all future requests are known to the system in advance. The assumption was unrealistic in the sense that it was not representative of the real world cloud computing environments which are inherently dynamic in nature. Considering this one might argue, why we wasted time and a complete chapter on the static problem if it was based on an unrealistic assumption. The reason is, while solving a linear program using an LP solver may be trivial, formulating a problem isn’t. The assumptions we make for the dynamic problem are realistic, but the linear program that it will eventually solve is theoretically the same as formulated in the static problem. Also, solving the static problem gives us a baseline to compare the performance of dynamic problem with, which is typically done to obtain a competitive analysis.

Solving the dynamic problem is much more challenging than solving the static one, as requests can arrive at any time and the network characteristics can change at any time. Let’s explain this in more detail, as we see in Section 3.3.2 a typical problem with hundreds of datacenter nodes, thousands of edges, tens of thousands of requests and hundreds of time units will result in a linear program with millions of variables and constraints. A trivial solution for solving the dynamic problem wherein the system has no knowledge of future demands would be to solve the proposed linear program every time a request arrives. However, solving a linear program with millions of variables and constraints in every timeslot would be extremely burdensome computationally. Hence, to solve the dynamic problem we need to design heuristics which make the problem feasible for the current traffic engineering systems while meeting Vritti’s design goals.
In this chapter we first describe the design goals and system model, followed by a concrete description of the dynamic problem. We then describe the control knobs in the system, followed by algorithm design. We propose four heuristics to solve the dynamic problem. We then evaluate the proposed algorithms using a simulation.

4.1 Design Goals and System Model

In this section we discuss the design goals for Vritti for solving the dynamic problem. We also discuss the system model in terms of traffic, network, revenue and fairness.

The design goals of Vritti for solving the dynamic problem in a single cloud environment are the same as that for the static problem (see Section 3.1) i.e., maximizing provider revenue, providing fairness and providing a control knob for revenue vs. fairness trade-off.

It is important to note that, at the core, the underlying problem Vritti solves is to provide guarantees for mix-flows in inter-datacenter wide area networks in single cloud computing environments, which is the same for both the static and the dynamic problem. We are aware that assuming a dynamic cloud computing environment for solving the dynamic problem, opens up the opportunity for considering more design goals, such as fault tolerance etc., however, that is out of scope for this dissertation.

In Chapter 3, we solved the static problem to mainly meet the first design goal i.e., maximizing provider revenue. If we refer back to the equation 3.4, we will see the wt\textsubscript{fair} factor, was set to one while solving the static problem. It made more sense to update the control knob wt\textsubscript{fair} dynamically as we discuss in Section 3.2.4. Hence, we deferred meeting the remaining design goals to this chapter. However, in Chapter 3, for completeness, we did specify our intuition behind outlining the three design goals and proposed approaches for meeting the design goals of providing fairness and providing control knob for revenue vs. fairness trade-off (see Section 3.2.4). In this chapter we build heuristics based on the proposed approaches to meet all the three design goals. The solutions are built taking into consideration the inherent dynamic nature of the problem.

The WAN traffic model remains the same as described in Section 3.2.1 of the static problem, with the only difference being that the system becomes aware of a request, only at the time of request arrival, not before then. The traffic specification and definition of flow types also remain the same. Also, the underlying network is the same as described in Chapter 3.

The revenue model and fairness model for the dynamic problem are also the same as defined for the static problem.
4.2 Dynamic Problem

In this section we provide a formal description of the dynamic problem, and specify the main control knobs that must be tuned to solve the problem.

4.2.1 Exact Problem Statement

Given an inter-datacenter network connecting datacenters in a multi-tenant environment, also given transfer requests by each tenant with requests arriving in an online manner, meaning, the system is aware of the request only when it arrives.

The objective is to identify the admission control, scheduling and routing decisions for five request types such that all guarantees are met and revenue is maximized. Admission control, scheduling and routing are the control knobs that determine the flow allocation.

This is a considerably complex problem due to its online nature. As discussed earlier, solving the static (offline) problem every time a request arrives is computationally infeasible. The problem is all the more challenging for us given we handle mix-flows, and provide guarantees to all request types, which has not been done before.

4.2.2 Control Knobs

The main control knobs of the problem are the same as static problem i.e., admission control, scheduling and routing. This is because we are solving the dynamic problem to generate admission control, scheduling and routing decisions, in a dynamic environment instead of a static one.

However, by the virtue of solving the problem in a dynamic environment, there is an additional control knob that comes into play. As done in previous works such as Amoeba [53], Tempus [29], PGA [57], Pretium [41] and in production inter-datacenter wide area networks such as G-Scale by Google [26] and IDN by Microsoft [25], time is discretized into timeslots, usually ranging from 3 to 5 minutes. This means that the SDN controller makes decisions at timeslot-level granularity.

In this work, we group timeslots into windows, and the size of the window determines the size of the linear program we solve (see calculation of number of constraints and variables in Table 3.4). We discuss this in more detail in the next section. Hence, window size is the additional control knob.
We play with this knob when evaluating the system for parameter sensitivities in Section 4.4.3.

This additional control knob allows the provider to decide how far in the future they would want to plan request allocation at a time. If computational feasibility is not a constraint for the provider, window sizes can be much longer. However, as with any control knob, with additional flexibility, we get the overhead of tuning the knobs. But, once the knob is tuned, and its values decided based on system environment considerations, it is a tool in the hands of the cloud provider.

4.3 Algorithm Design

In this section we propose the heuristics to solve the dynamic problem. In addition to meeting the design goals, we also want the algorithms to be computationally feasible.

As discussed in Section 4.2.2, we use the concept of time window to reduce the problem size, where each window comprises of a few timeslots. If we were to run the static problem proposed in Chapter 3 for every timeslot for the total system run time, the problem would become extremely large in size, for it would have to plan for all incoming requests along all future timeslots. So, instead of planning for the entire duration of system run time, we break the problem by planning for a window at a time i.e., an LP is run for each timeslot with the window end time as the total run time, thus breaking the problem into a considerably smaller size. We use this method in all the proposed algorithms for computational feasibility.

We first propose a greedy algorithm that allocates requests in a greedy manner when they arrive, with the objective of maximizing the minimal fraction of soft deadline requests, and maximizing overall revenue. However, with the greedy algorithm we only aim to meet the first design objectives of maximizing revenue, we extend the greedy algorithm with a fairness heuristic to meet the second and the third design objectives of providing fairness and providing a control knob for revenue vs. fairness trade-off.

By extending the greedy algorithm with a fairness heuristic, we aim to meet all the three design objectives. However, we believe that the greedy solution can get stuck in a local optima because of its greedy nature of allocating requests. To avoid that and to improve the revenue generated, we propose a selective-rescheduling heuristic. We extend the selective-rescheduling algorithm by adding a fairness objective to provide fairness to non-deadline requests while minimally trading-off revenue due to deadline requests. We describe the algorithms in more detail in sections below.

The greedy-fair and selective-rescheduling algorithms are more complex as compared to greedy, as they have to run additional routines besides the LP. Hence, we trade-off computational time for more fairness and higher revenue respectively. In the case of selective-
rescheduling-fair algorithm, we trade-off higher computational time for both higher fairness and higher revenue.

4.3.1 Greedy Algorithm

In this section we describe the greedy algorithm in terms of its objectives and formulate the greedy solution to the dynamic problem in Algorithm 1.

Algorithm Description

With the greedy algorithm our main objective is to provide a straight-forward approach to solve the dynamic problem to allow computational feasibility while meeting mix-flow guarantees and maximizing revenue.

The algorithm allocates a request in space and time such that its guarantees can be met and provider revenue is maximized. Any request which cannot be provided its required guarantee is rejected for good, unless the tenant resubmits the request. For all requests that arrive in a timeslot, the requests are split proportionally into chunks such that they fit into the current time window. The remainder of each request is postponed to next time window. Then the two LPs are solved one after the other. By solving the first linear program we obtain max-min fraction to be guaranteed to all type 2 and 3 requests i.e., soft deadline requests. By solving the second linear program, we obtain the scheduling and routing decisions that maximize the revenue due to all the requests that arrived in a timeslot. Algorithm 1 describes the complete greedy algorithm.

To make the problem computationally feasible, we need to reduce the problem size. To do that, instead of planning scheduling and routing decisions for all future timeslots, we plan for a few timeslots at a time i.e., one window at a time. We are able to implement that by splitting each new request when it arrives, proportionally such that it fits a time window. Which means, if the hard deadline of a request is ten timeslots after its arrival timeslot, window size is five timeslots, and the request demands is 100 GB. The request will be split into two chunks, since we have two time windows, the volume will be split proportionally, 50 GB for each window. The second request is allocated at the beginning of the second time window.

We are aware that there can be multiple ways to split a request; however, for simplicity, we choose to proportionally divide the requests. We do not take into account the expected network utilization during a time window when splitting the request. The only way to do that would have been to calculate link utilization for all links along all candidate paths between request arrival time and request deadline. Based on the average link utilization for a path for a time window, we could have split the request. For example, if average link utilization for a candidate path is 60% for the current time window, and 20% for the next time window, we can split the
request demanding transfer of volume 100 GB such that 40% of the request is allocated in the current time window and remaining in the next time window. However, we cannot know the network utilization of a future time window, as we only plan for the current time window at a time. Even if we were able to estimate the network utilization of a future window, we must also consider that a request can be split based on the link utilization along all candidate paths for a request, hence, extensively increasing the computational complexity. In reality, it is up to the LP solver to decide which path(s) are used to route a request, and we can never certainly estimate future path utilization until the path(s) is/are selected. Hence, we do not believe we are paying a large price by proportionally splitting requests among future time windows.

In terms of providing guarantees to mix-flows, we must note that the guarantees for deadline requests are formulated as constraints in the LP formulation. Hence, if an LP is able to find a feasible solution in a timeslot, it means it is able to meet all the constraints and also the guarantees. Else, the requests in the timeslot are rejected. If a request has deadline larger than the window size, the entire request may not be rejected if it is a non-hard deadline request, as the portion of the request queued in the future window is yet to be solved for.

With this greedy approach we greedily try to maximize revenue due to all flow types by allocating partial requests between the current timeslot and end time of the current window. We do not provide fairness to non-deadline flows, and we do not tune the control knob to allow fairness vs. revenue trade-off. We propose a greedy algorithm with fairness objective to meet these shortcomings in the next section.

Algorithm 1 Greedy Algorithm.

**Input:** Network Topology: \( G(N,E) \), \( y_k(t,i) = \{ S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_1,t,i}, k_{d_2,t,i}, V_{t,i}, P(t,i) \} \). where \( y \) is a request generated in timeslot \( k \).

\( Y_{k,all} = \{ y_{1,1}, y_{1,2}, ..., y_{i,i}, ..., y_{t,i} \} \), all tenant requests generated in timeslot \( k \).

**Output:** Admission, Scheduling and Routing

1: for Requests arrived in time slot \( k \) of window \( w \) do
2: Split the request \( (V_{t,i}, k_{d_1,t,i}, k_{d_2,t,i}) \) s.t. it fits in \( w \)
3: Postpone the remainder of the request to \( w + 1 \)
4: \( \max \min \alpha_{t,i} \)
5: \( \max R(x) \quad \forall x \in X_q \)

4.3.2 Greedy Algorithm with Fairness Objective

In this section we extend the greedy algorithm described above to provide fairness for non-hard deadline requests i.e., type 2, type 4 and type 5 requests. We describe the objectives of the
Algorithm Description

The objective of this algorithm is to provide fairness to non-hard deadline requests i.e., type 2 requests with soft deadlines, type 4 and type 5 requests with no deadlines, in addition to providing guarantees to mix-flows and maximizing provider revenue.

The algorithm allocates requests in space and time such that their guarantees can be met, provider revenue is maximized and fairness is provided to type 2, 4 and 5 requests. For all requests that arrive in a timeslot, the requests are split proportionally into chunks such that they fit into the given window size. The remainder of each request is postponed to next time window. We use a sliding window approach instead of fixed windows that were used in the greedy algorithm. Then the two LPs are solved one after the other. By solving the first linear program we obtain max-min fraction to be guaranteed to all type 2 and 3 requests i.e., soft deadline requests. By solving the second linear program, we obtain the scheduling and routing decisions that maximize the revenue due to all the requests that arrived in the timeslot.

When either of the two LPs became infeasible in a timeslot in the greedy approach, all the pending requests in the timeslot were rejected. But in this approach, whenever the LPs become infeasible, we relax the constraints, and solve the LP again. If the LP becomes infeasible again, we move on to the next timeslot. We state the LP with the updated constraints in equations 4.1 to 4.8. The demand constraints for type 4 and 5 requests (equations 4.5, 4.6), the capacity and non zero constraints (equations 4.7, 4.8) remain unchanged. We however enforce zero allocation for type 1 and type 3 requests with constraints in equations 4.2 and 4.4 respectively. We remove the constraint representing partial guarantee for type 2 requests by the soft deadline. Algorithm 2 describes the complete algorithm.

In terms of making the problem computationally feasible, like the greedy algorithm, we split the request sizes and only find allocations for the size of windows. Hence, the problem size is reduced.

\[
\max R_{fair}(x) \quad \forall x \in X_q
\] (4.1)
s.t.

\[ \sum_{t,i \in \text{type } 1 \text{ req. } k=k_{s,t,i}} \sum_{p \in P_{t,i}} x(t,i,p,k) = 0 \ (Demand - Type1) \] (4.2)

\[ \sum_{t,i \in \text{type } 2 \text{ req. } k=k_{s,t,i}} \sum_{p \in P_{t,i}} x(t,i,p,k) \leq V_{t,i} \ \forall t,i \ (Demand - Type2) \] (4.3)

\[ \sum_{t,i \in \text{type } 3 \text{ req. } k=k_{s,t,i}} \sum_{p \in P_{t,i}} x(t,i,p,k) = 0 \ \forall t,i \ (Demand - Type3) \] (4.4)

\[ \sum_{t,i \in \text{type } 4 \text{ req. } k=k_{s,t,i}} \sum_{p \in P_{t,i}} x(t,i,p,k) \leq V_{t,i} \ \forall t,i \ (Demand - Type4) \] (4.5)

\[ \sum_{t,i \in \text{type } 5 \text{ req. } k=k_{s,t,i}} \sum_{p \in P_{t,i}} x(t,i,p,k) \leq V_{t,i} \ \forall t,i \ (Demand - Type5) \] (4.6)

\[ \sum_{t} \sum_{i} \sum_{e \in p \in P_{t,i}} x(t,i,p,k) \leq c(e,k) \ \forall e,k \ (Capacity) \] (4.7)

\[ x(t,i,p,k) \geq 0 \ \forall t,i,p \in P_{t,i}, k \in [k_{s,t,i}, \min(k_{d_2,t,i}, K)] \] (4.8)

Unlike the greedy algorithm, we use a sliding window approach when allocating a request. Which means scheduling and routing decision for all the requests that arrive in a timeslot \( k \) are made from timeslot \( k \) to timeslot \( k + w_{\text{size}} \) where \( w_{\text{size}} \) is the size of the window. In the greedy algorithm, we divided time into static windows, irrespective of the request arrival time. Assuming window size of five, each new window would begin at static timeslots, 0, 5, 10, 15 . . . and so on. Consequently, a request arriving at timeslot 0, will have five timeslots to complete, while request arriving at timeslot 2 will have only three timeslots. Hence, in the greedy algorithm, requests arriving later in a window have relatively lesser time to complete. So, we use a sliding window approach instead.

The guarantees for deadline requests are formulated as constraints of the problem. For non-deadline requests, we enforce their allocation when deadline requests cannot be met. But we do not guarantee their completion. Here, the objective of maximizing provider revenue should be considered with a grain of salt, since we believe we cannot achieve fairness without trading off revenue. But we can also not simply forego revenue as an objective, as there has to be realistically some incentive for the provider to implement this approach. Hence, the main knob that the provider can tune is deciding when to use the updated LP for allocating non-deadline requests.

In order to provide fairness to non-hard deadline requests, we accommodate type 2, 4, and 5
requests whenever the LP is infeasible. It can be very difficult to identify which request caused the LP to become infeasible. The hard deadline requests have the strictest constraints, they are allocated with an all or nothing approach to meet the deadlines. A hard deadline request has to be completely met, there is no point in allocating the partial demand of such a request, hence once a hard request is rejected we run the updated LP to allocate that capacity to non-hard deadline and non-deadline requests. Hence we make two main trade-offs, first, once a hard deadline request is rejected, it is rejected for good, hence, we trade-off revenue due to hard deadline request by not giving such requests a second chance. Second, in every timeslot the LP is infeasible, we effectively run two additional LPs in that timeslot, hence, we trade-off computation time for increased fairness.

The algorithm is as computationally feasible as the greedy algorithm, because the problem size in terms of the LPs remains the same as that of greedy. We believe the increase in complexity due to an additional fairness calculation is trivial, as it is a heuristic that is run every $k$ timeslots, that does not modify the LP in terms of number of objectives and constraints. The algorithm provides guarantees to all mix-flows in the same way as the greedy algorithm does.

It is important to note, that while the algorithm checks all boxes in terms of design goals, once it rejects a hard deadline request, it is rejected for good. We believe that there can be scope of improving the revenue further by rescheduling initially rejected requests. We propose a selective-rescheduling algorithm in the next section to do so.

---

**Algorithm 2 Greedy Fair Algorithm.**

**Input:** Network Topology: G(N,E), $y_k(t,i) = \{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_1,t,i}, k_{d_2,t,i}, V_{t,i}, P(t,i)\}$; where $y$ is a request generated in timeslot $k$.

$Y_{k,all} = \{y_{1,1}, y_{1,2}, ..., y_{1,i}, ..., y_{t,i}\}$, all tenant requests generated in timeslot $k$.

**Output:** Admission, Scheduling and Routing

1: for Requests arrived in time slot $k$ do
2: Split the request $(V_{t,i}, k_{d_1,t,i}, k_{d_2,t,i})$ s.t. it fits in $w$
3: Postpone the remainder of the request to $k + w_{size}$
4: max min $\alpha_{t,i}$
5: if max min $\alpha_{t,i}$ is infeasible then
6: max $R_{fair}(x)$
7: else
8: max $R(x)$ \hspace{0.5cm} $\forall x \in X_q$
9: if max $R(x)$ is infeasible then
10: max $R_{fair}(x)$

---

83
4.3.3 Selective Rescheduling Algorithm

The selective-rescheduling algorithm selectively reschedules already rejected requests to improve overall request acceptance rate. In this section we describe the approach and formulate the algorithm.

Algorithm Description

The idea behind this approach is to accommodate already rejected requests. It can be argued that if a request is rejected once, how can we guarantee that it is not rejected after rescheduling. The short answer is, we cannot. Before we delve deeper, it is important to understand why a request or a set of requests can be rejected by the controller. Whenever an LP solver is not able to find a feasible solution for a set of requests in a timeslot, it will be mainly because it is not able to meet all the constraints. This means that if, in a current set of requests that are inputs to the LP, exists a hard deadline request that cannot be met, the entire set of requests in the timeslot will be rejected as the LP becomes infeasible. Similarly, if the capacity constraint is violated i.e., no set of paths exist to allocate all requests without over-provisioning link capacity, the LP becomes infeasible. It can be very difficult to identify the main culprits in the set of requests that cause the LP to become infeasible, hence, we try to selectively reschedule some of the rejected requests in future timeslots. We cannot claim certainly that the rescheduled request will be allocated, as we cannot predict future requests in a realistic cloud computing environment, and as a consequence, we do not have full control over the input being generated for the LP in a timeslot. However, by selecting a few rejected requests and rescheduling them in a future timeslot, we increase the chance of the rejected request to be allocated. Hence, increasing the chance of improving revenue and request acceptance rate. Additionally, instead of planning the schedule and route of hard deadline requests one window at a time, we plan for the entire duration of the request. We do this to provide true apriori guarantee for hard deadline requests.

In this algorithm, the non-hard deadline requests are proportionally split such that they fit in the given window size. Like the greedy-fair algorithm, we use a sliding window approach, postponing the remainder of the request after splitting to the next time window. We then solve the linear programs to maximize the minimal fraction guaranteed for soft deadline requests, and to maximize the revenue. We do not split the hard deadline requests. When a hard deadline request arrives, we solve the linear programs to maximize the minimal fraction guaranteed if it is a type 3 request, and to maximize the revenue to identify schedule and route for the request from request arrival time till the hard deadline if it is either type 1 or type 3 request. If a hard deadline request is rejected it is added to the hard deadline rejection queue. If soft or non-deadline request is rejected it is added to a non-hard deadline rejection queue. The rejection
queues are updated every timeslot, wherein all the requests with expired deadlines are removed. In every timeslot, we first check if the hard deadline queue is empty, if it is not, we select $j$ previously rejected requests with highest throughput requirement and add them to the current timeslot. If the hard deadline queue is empty, we select $j$ previously rejected requests from the non-hard deadline rejection queue with highest throughput requirement and add them to the current timeslot. For each of the $j$ selected requests, we make sure that the deadline of the request is greater than $k + w_{\text{size}}$ where $k$ is the current timeslot, and the request has not been rejected more than $n$ times. We take these measures to avoid overwhelming the solver with extremely high traffic demands and to avoid unfairly giving too many chances to the rejected requests, respectively. We assume sizes of the rejection queues are infinite.

To make the problem computationally feasible, we split all the non-hard deadline requests and non-deadline requests, hence, reducing the problem size. We trade-off computation time to provide true apriori guarantees for hard deadline requests.

In this algorithm, we introduce two control knobs for the cloud provider, namely, maximum number of previously rejected requests $j$ to be rescheduled and number of times a request is allowed to be rescheduled before it is finally rejected $n$. The intuition behind providing these control knobs is providing flexibility in the hand of the cloud provider. With number of rejected requests to be rescheduled in a timeslot i.e., $j$ the provider can decide based on the request arrival rate and network capacity, how many rejected requests it can accommodate in a timeslot. Similarly, with number of times a request can be rescheduled i.e., $n$, the providers can decide on the basis of average duration of the request and average network utilization, the number of times a rejected request is allowed to be rescheduled.

We clearly state that with selective-rescheduling we cannot guarantee a rescheduled rejected request will be allocated. This is attributed to the randomness in the input requests generated at every timeslot. However, when the algorithm is run for sufficiently long time, we believe, again attributing to unpredictable future requests, that by rescheduling rejected requests, we increase the odds of the request being allocated in a future timeslot.

With this algorithm, we trade off additional complexity and computation time for a higher revenue. Moreover, we trade-off bandwidth allocation to non-deadline and soft deadline flows for higher acceptance rate.

4.3.4 **Selective Rescheduling Algorithm with Fairness Objective**

The selective-rescheduling algorithm with fairness objective selectively reschedules already rejected requests to improve overall request acceptance rate, and average bandwidth allocated to non-deadline requests. In this section we describe the approach and formulate the algorithm.
Algorithm 3 Selective Rescheduling Algorithm.

**Input:** Network Topology: $G(N,E)$, $y_k(t,i) = \{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_1,t,i}, k_{d_2,t,i}, V_{t,i}, P(t,i)\}$; where $y$ is a request generated in timeslot $k$.

$Y_{k,all} = \{y_{1,1}, y_{1,2}, \ldots, y_{1,i}, \ldots, y_{t,i}\}$, all tenant requests generated in timeslot $k$.

**Output:** Admission, Scheduling and Routing

1: for Requests arrived in time slot $k$ do
2: Split type 2, 4 and 5 requests ($V_{t,i}, k_{d_1,t,i}, k_{d_2,t,i}$) s.t. it fits in $w$
3: Postpone the remainder of each request to $k + w_{size}$
4: $\max \min \alpha_{t,i}$
5: $\max R(x)$ $\forall x \in X_q$
6: if request not allocated then
7: Add request to $Q_{rr}$
8: Select $j$ type 1,3 rejected requests with the highest throughput demand from $Q_{rr}$
9: if no hard deadline request in $Q_{rr}$ then
10: Select $j$ type 2,4,5 rejected requests with the highest throughput demand from $Q_{rr}$
11: Postpone the selected $j$ requests to next timeslot
12: Pop the selected $j$ requests from $Q_{rr}$

**Algorithm Description**

With this algorithm, the objective is to achieve the benefit of higher acceptance rate due to rescheduling deadline requests using selective-rescheduling algorithm, without considerably trading off the bandwidth allocated to non-deadline requests, the main trade-off of selective-rescheduling algorithm.

In selective-rescheduling-fair algorithm, like selective-rescheduling we only split soft deadline and non-deadline requests. We reschedule rejected requests, giving higher priority to deadline requests over non-deadline requests. But if the linear program(s) are infeasible, we solve the updated LP in equation 4.1 with relaxed constraints to allocate non-deadline requests without the more burden-some deadline requests, like the greedy-fair algorithm . Like all the algorithms described before, we solve two linear programs to allocate requests; one, for fairness among soft deadlines i.e., maximizing the minimal fraction to be guaranteed for soft deadline requests, second to maximize overall revenue. Similar to greedy-fair algorithm, if any of the two linear programs are infeasible in a timeslot, we relax the constraints and solve the linear program to maximize revenue due to only non-deadline requests. If still, the linear program is infeasible, all the requests in the timeslot are rejected and put in the rejection queue. At the end of every timeslot henceforth, the rejection queue is checked and $j$ requests with the highest throughput requirement are selected and rescheduled in the next timeslot. Hard deadline requests are given higher priority over non-deadline requests for rescheduling. Only, if there are no hard deadline requests, non-deadline requests are selected for rescheduling.
To make the problem computationally feasible, we split all the non-hard deadline requests and non-deadline requests, hence, reducing the problem size. We trade-off computation time to provide true apriori guarantees for hard deadline requests.

The main control knobs are the number of requests selected for rescheduling and deciding when the constraints of the LP must be relaxed. The former control knob allows the provider to decide how many rejected requests must be rescheduled, which mainly depends on the expected load on the network. The latter depends on how much provider is ready to trade-off revenue due to deadline requests in lieu of revenue due to non-deadline requests. This knob is another tool in the hand of the cloud provider to decide which requests must be allocated to account for their revenue. In this work, we give a clear preference to deadline requests, by allocating non-deadline requests only when linear programs become infeasible. It is more likely that the LPs become infeasible due to deadline requests which have the strictest deadlines. A provider can periodically relax the constraints to meet non-deadline requests if they can incur the consequent trade-off.

Like selective-rescheduling, in selective-rescheduling-fair as well we cannot guarantee a rescheduled rejected request will be allocated. Unlike selective-rescheduling, by solving the updated LP whenever a request is not allocated, we can however increase the bandwidth allocated to non-deadline requests.

We are trading off higher computational complexity by adding two additional components, mainly the rescheduling component and re-running of the linear programs.

### 4.3.5 Discussion

One can argue, why we proposed four different heuristics, each with its own control knobs, and why not one cumulative heuristic that could meet all the objectives and tune all the control knobs. Our algorithm design process involved developing a fundamental algorithm that solved the LPs, keeping them computationally feasible, and building more heuristics from there. The idea is to present to the reader, the entire algorithm design process, and how a new algorithm is derived from another by tuning the design trade-offs.

### 4.4 Simulation and Evaluation

In this section we evaluate the performance of the four algorithms proposed to solve the dynamic problem.
Algorithm 4 Selective Rescheduling Algorithm with Fairness Objective.

**Input:** Network Topology: $G(N,E)$, $y_k(t,i) = \{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_1,t,i}, k_{d_2,t,i}, V_{t,i}, P(t,i)\}$; where $y$ is a request generated in timeslot $k$.

$Y_{k,all} = \{y_{1,1}, y_{1,2}, ..., y_{1,i}, ..., y_{t,i}\}$, all tenant requests generated in timeslot $k$.

**Output:** Admission, Scheduling and Routing

1. for Requests arrived in time slot $k$ do
2. Split type 2, 4 and 5 requests ( $V_{t,i}, k_{d_1,t,i}, k_{d_2,t,i}$) s.t. it fits in $w$
3. Postpone the remainder of each request to $k + w_{size}$
4. max min $\alpha_{t,i}$
5. if max min $\alpha_{t,i}$ is infeasible then
6. max $R_{fair}(x)$
7. else
8. max $R(x) \forall x \in X_q$
9. if max $R(x)$ is infeasible then
10. max $R_{fair}(x)$
11. if request not allocated then
12. Add request to $Q_{rr}$
13. Select $j$ type 1,3 rejected requests with the highest throughput demand from $Q_{rr}$
14. if no hard deadline request in $Q_{rr}$ then
15. Select $j$ type 2,4,5 rejected requests with the highest throughput demand from $Q_{rr}$
16. Postpone the selected $j$ requests to next timeslot
17. Pop the selected $j$ requests from $Q_{rr}$
4.4.1 Simulation Environment and Performance Metrics

The simulation environment is same as that of the static problem (see Section 3.4.1), with one main difference - time is discretized into timeslots, where each timeslot is five minutes long, and five timeslots constitute one time window. To the best of our knowledge, all of the recent work in the area has assumed one timeslot to be equivalent to five minutes, and we do the same to remain consistent. We run the simulation for 30 timeslots, 30 times. We chose 30 timeslots, mainly due to high compute capacity required. We ran the experiments on the compute clusters available through NC State University’s High-Performance Computing (HPC) Services [67].

The performance metrics remain the same as that of the static problem namely (see Section 3.4.1), average request acceptance rate, average fraction guaranteed, average fraction of type 4 requests allocated, average fraction of type 5 requests allocated, average network utilization, average throughput, average revenue, time taken and memory usage.

We evaluate all the metrics with respect to request arrival rate, mainly because with request arrival rate, we can evaluate the algorithms both in lightly loaded and heavily loaded scenarios. We model the request arrival process as a Poisson process with parameter $\lambda \in [1,10]$. The parameter space to evaluate the metrics is huge; we explore this further when evaluating the algorithms to understand parameter sensitivities.

4.4.2 Top Level Questions

In this section we pose the following specific questions. We answer these questions through exhaustive evaluation.

1. **How effective is the algorithm in providing guarantees to mix-flows?** - With this question, we aim to analyze the effectiveness of the four algorithms in providing guarantees to mix-flows - namely, providing complete transfer guarantee for hard deadline requests i.e., type 1 and type 3 requests, providing partial transfer guarantee for soft deadline requests i.e., type 2 and type 3 requests, providing fairness for non-deadline requests i.e., type 4 and type 5 requests by increasing the fraction of their transfer completion by the system run time.

   - Type 1 and 3 requests with Greedy, GF, SR and SRF - How does request acceptance rate vary with request arrival rate?
   - Type 2 and 3 requests with Greedy, GF, SR and SRF - How does average fraction guaranteed vary with request arrival rate?
   - Type 4 and 5 requests with Greedy, GF, SR and SRF - How effective is the algorithm in providing fairness to mix-flows? How does average fraction allocated for type 4 and type 5 requests vary with request arrival rate?
2. **How does network utilization vary with request arrival rate?** - With this question, we aim to evaluate the algorithms with respect to average network utilization in lightly loaded and highly loaded conditions. A good algorithm should be able to effectively provide guarantees to mix-flows and be able to effectively utilize the network capacity in doing so.

3. **How does throughput vary with request arrival rate?** - With this question, we aim to evaluate the algorithms with respect to average throughput in lightly loaded and highly loaded conditions.

4. **How does greedy-fair algorithm perform w.r.t. different fairness weights?** - With this question, we aim to evaluate the performance of the algorithm w.r.t fairness weight $w_{\text{fair}}$. Fairness weight is designed as a control knob for flexibility in the hands of the cloud provider in providing fairness.

5. **Assuming a fixed pricing function, how does revenue vary with request arrival rate?** - Assuming one unit of revenue is associated with one Gb/timeslot transmitted, we aim to evaluate the performance of the algorithms w.r.t. revenue accrued.

6. **How does space and time overhead of the algorithms vary with request arrival rate?** - With this question, we aim to evaluate the performance of the algorithms w.r.t. allocation time and memory usage.

7. **How do the algorithms perform w.r.t. other parameters?** - With this question, we aim to evaluate the algorithms w.r.t. window size and tighter deadlines.

### 4.4.3 Simulation Results

In this section, we evaluate the results of the simulation to answer the top level questions proposed.

**Q1. How effective is the algorithm in providing guarantees to mix-flows?**

- **Type 1 and 3 requests with Greedy, GF, SR and SRF** - We evaluate the allocation of hard deadline requests i.e., type 1 and type 3 requests with percentage request acceptance rate w.r.t. arrival rate. A high acceptance rate signifies a larger percentage of hard deadline requests were fully completed by the hard deadline, and a lower acceptance rate signifies the contrary. As we can see in Figure 4.1, the acceptance rate decreases with increasing arrival rate. As the arrival rate increases, the system is more heavily loaded, and the contention for network capacity increases, hence the decrease in acceptance rate.
As we can see in Figure 4.1, selective-rescheduling and selective-rescheduling-fair clearly outperform greedy and greedy-fair algorithms with consistently higher acceptance rate. This is mainly because with selective-rescheduling and selective-rescheduling-fair, we plan for the entire duration of hard deadline requests when the request arrives, instead of splitting the hard deadline requests and planning for a window at a time like we do with the other two algorithms. By splitting hard deadline requests and reserving network capacity spatially and temporally only for the duration of window size, the greedy and greedy-fair algorithms can not truly guarantee transfer completion until the beginning of the last window for the request. As a consequence, the hard deadline requests have lower acceptance rate when contention for network capacity increases at higher arrival rate. Additionally, selective-rescheduling and selective-rescheduling-fair also reschedule requests when they are rejected wherein deadline requests are given clearly higher priority, hence further increasing the odds of hard deadline requests getting accepted. Selective-rescheduling also outperforms selective-rescheduling-fair, because selective-rescheduling-fair employs a fairness component that accommodates non-deadline requests whenever any of the linear programs become infeasible. Both the greedy-fair and greedy algorithms, treat hard deadline requests in a similar fashion, with one main difference. Greedy-fair algorithm uses a sliding window approach when allocating request, hence, acceptance rate is similar for the two algorithms.

Figure 4.1: Acceptance Rate for Type 1 and Type 3 Flows in G-Scale.
- **Type 2 and 3 requests with Greedy, GF, SR and SRF** - We evaluate the allocation of soft deadline requests i.e., type 2 and type 3 requests with average fraction guaranteed w.r.t. arrival rate. A high average fraction guaranteed signifies on an average, for soft deadline requests, a larger fraction of demand requested was completed by the soft deadline, and a lower average fraction guaranteed signifies the contrary. As we can see in Figure 4.2, the average fraction guaranteed decreases with increasing arrival rate. As the arrival rate increases, the system is more heavily loaded, and the contention for network capacity increases, hence the decrease in average fraction guaranteed. Percentage average fraction guaranteed is expected to be low because soft deadline requests have looser constraints as compared to hard deadline requests. With higher arrival rate selective-rescheduling and selective-rescheduling-fair observe a sharper decline as compared to greedy and greedy-fair algorithms. This is mainly due to the explicit higher priority given to hard deadline requests by selective-rescheduling and selective-rescheduling-fair, in terms of both long term reservation and rescheduling of requests. Both greedy and greedy-fair algorithms treat soft deadline requests in a similar fashion, and we can observe that in their similarly behaving plots.

![Figure 4.2: Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale.](image)

- **Type 4 and 5 requests with Greedy, GF, SR and SRF** - We evaluate the allocation of non-deadline requests i.e., type 4 and type 5 requests with average fraction allocated
w.r.t. arrival rate. A high average fraction allocated signifies on an average, for non-
deadline requests, a larger fraction of demand requested was completed, and a lower
average fraction allocated signifies the contrary. As we can see in Figures 4.3 and 4.4, the
average fraction allocated decreases with increasing arrival rate for selective-rescheduling-
fair, greedy and selective-rescheduling, however remains close to 100% for greedy-fair
algorithm. This is because, unlike greedy and selective-rescheduling, greedy-fair algorithm
allocates non-deadline requests whenever any of the linear programs become infeasible.
While selective-rescheduling-fair also allocates non-deadline requests whenever any of the
linear programs become infeasible, it gives higher priority to hard deadline requests by
reserving bandwidth for them as and when the requests arrive and selectively rescheduling
hard deadline requests with higher priority. Thus, selective-rescheduling-fair does not fare
as well as greedy-fair, but performs better than selective-rescheduling and greedy. Since
selective-rescheduling is designed with objective of improving acceptance rate of hard
deadline requests with an all-or-nothing approach, it trades off bandwidth allocated to
non-deadline requests for hard deadline requests and sees the steepest drop for fraction
allocated for type 4 and type 5 requests. Hence, with minimal drop in acceptance rate, with
greedy-fair algorithm we are able to maximize the allocation to non-deadline requests.

![Figure 4.3: Average Fraction Allocated for Type 4 Flows in G-Scale.](image-url)
Q2. How does network utilization vary with request arrival rate?

Network utilization increases with request arrival rate as can be seen in Figure 4.5. This is expected with an increase in arrival rate. However, selective-rescheduling-fair, greedy and selective-rescheduling saturate at 65%, 63% and 53% respectively, while greedy-fair outperforms the three algorithms with network utilization as high as 96% at arrival rate 10. This is because, greedy-fair utilizes the capacity of the network whenever any of the linear programs become infeasible, by relaxing constraints, and allocating non-deadline requests along space and time which are not associated with any strict constraints. Hence, greedy-fair algorithm is able to accommodate the most traffic, clearly outperforming selective-rescheduling-fair, selective-rescheduling and greedy when it comes to effectively utilizing the network by accommodating non-deadline requests with little drop in acceptance rate for hard deadline requests.
Q3. How does throughput vary with request arrival rate?

Average per job throughput calculated in Gb per timeslot decreases with request arrival rate as can be seen in Figure 4.6. This at first may not seem intuitive, but in this simulation the types of incoming requests are selected from a uniform distribution of five types. Hence, probabilistically, half of the incoming requests are not associated with any deadlines and hence, have lower throughput requirements. This dampens the throughput due to all the algorithms. The greedy algorithm has consistently lower per job throughput as compared to greedy-fair, selective-rescheduling-fair and selective-rescheduling. Greedy-fair achieves higher throughput for higher request arrival rate as compared to selective-rescheduling. This is mainly because, the non-deadline requests in greedy-fair algorithm contribute higher throughput as compared to hard deadline requests in selective-rescheduling. This can also be attributed to diversity of input request types. Hence, if we update the input to reflect larger number of hard deadline requests, we should expect higher throughput due to selective-rescheduling and selective-rescheduling-fair as compared to greedy-fair.
Q4. How does greedy-fair algorithm perform w.r.t. different fairness weights?

We designed a fairness weight knob $w_{fair}$ when defining revenue in equation 3.4 in Section 3.2.3. The objective was to provide a knob in the hand of the cloud provider to tune the allocation of non-deadline requests flexibly by using the knob. However, as can be seen in Figures 4.7, 4.8, 4.9 and 4.10, contrary to the expectation, the allocation of deadline and non-deadline requests cannot be tuned using the fairness weight knob. This is because of all or nothing allocation approach we employed for allocating hard deadline requests. The flow allocation of a hard deadline request must always be 100% by their deadline, if no allocation is found, in greedy-fair we fix the flow allocation of hard deadline request to be 0%. This is due to the virtue of the guarantee we provide to hard deadline request i.e., the request must be completed before its hard deadline in order to be accepted. Hence, due to our design choice of providing this guarantee to hard deadline request, we cannot provide a dynamic fairness knob such as fairness weight. However, with the performance of the greedy-fair algorithm at all arrival rates for non-deadline requests, we provide a novel fairness solution that provides fairness with the least amount of price to be paid in terms of acceptance rate of hard deadline requests.
Figure 4.7: Acceptance Rate for Type 1 and Type 3 Flows in G-Scale.

Figure 4.8: Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale.
Q5. How does space and time overhead of the algorithms vary with request arrival rate?

As can be seen in Figure 4.11, time taken to generate spatial temporal allocation of requests is similar for all the algorithms. The time taken increases as the load on the network increases.
Selective-rescheduling and selective-rescheduling-fair take more time as compared to greedy and greedy-fair algorithms because of the rescheduling component and planning for longer term for hard deadline requests as compared to other request types. In terms of memory usage, as can be seen in Figure 4.12, memory usage by the greedy algorithm is the highest as compared to the remaining algorithms. This can be attributed to the use of static windows in the greedy algorithm and sliding windows in the other three algorithms. By using static windows, in the first timeslot in every window, a number of requests get queued up, and this number increases with time if requests have larger time duration and are split up uniformly so they can be completed as close to their deadline or total run time as possible.

![Figure 4.11: Time Taken in G-Scale.](image)
Q6. Assuming a fixed pricing function, how does revenue vary with request arrival rate?

Assuming one unit of revenue is associated with one Gb/timeslot transmitted, we aim to evaluate the performance of the algorithms w.r.t revenue accrued. Revenue increases with increasing arrival rate as can be seen in Figure 4.13, this is expected because the network is more heavily loaded at higher arrival rate to meet the increasing traffic demand. Revenue accrued by a request is a function of throughput and average path cost. Selective-rescheduling serves the highest number of hard deadline requests which have the highest contribution to overall revenue because of their higher throughput demand, hence we can see selective-rescheduling contributes most to overall revenue. selective-rescheduling-fair earns lower revenue because it is designed to trade-off revenue due to hard deadline requests for non-deadline requests. Greedy-fair compensates the revenue by accommodating a higher number of non-deadline requests, and hence at higher arrival rates contributes comparable revenue as selective-rescheduling-fair.
Q7. How do the algorithms perform w.r.t. other parameters?

**Impact of Window Size on Guarantees for Mix-flows** - To answer this question, we study the impact of window size on guarantees for mix-flows. We use window sizes 5 and 10 to do so, mainly because we run the algorithms for 30 timeslots, and increasing the window size larger than 10 would have been an overkill. The main difference that can be observed with using a larger window size consistently across the four algorithms is the average throughput, as can be seen in Figures 4.14, 4.15 and 4.16. This is because, with a larger window size the requests are split more coarsely, hence can be completed faster, comparatively. This is why in selective-rescheduling in Figure 4.16 we do not see any impact on throughput of increasing the window size to 10.
Figure 4.14: Average Throughput in G-Scale using Greedy.

Figure 4.15: Average Throughput in G-Scale using Greedy-fair.
Impact of Tightness of Deadlines on Guarantees for Mix Flows - We also study the impact of tightness of deadlines in meeting guarantees for mix-flows. Deadlines are emulated as an exponential process with mean of one hour or 12 timeslots in the remaining experiments. In this section, we study the impact on guarantees for mix-flows for deadlines calculated with mean 2, 4, 6, 8 and 10 timeslots. For hard deadline and non-deadline request, there was no observable impact due to tight deadlines. The performance of the algorithms w.r.t different deadlines was very similar to make any conclusive statement. However, for soft deadline, the average fraction guaranteed was lower for tighter deadlines and it increased as deadlines became looser, as can be seen in Figures 4.17, 4.18, 4.19, 4.20. We do not see any affect on hard deadline requests because we calculate the volume of all input requests as a function of throughput and deadline. As a result, tighter deadlines do not exert any pressure on the system when they are associated with hard deadline requests.
Figure 4.17: Average fraction guaranteed for type 2 and 3 flows in G-Scale with greedy algorithm with duration of request calculated as Exponential distribution with mean in \([2,4,6,8,10,12]\).

Figure 4.18: Average fraction guaranteed for type 2 and 3 flows in G-Scale with GF algorithm with duration of request calculated as Exponential distribution with mean in \([2,4,6,8,10,12]\).
Figure 4.19: Average fraction guaranteed for type 2 and 3 flows in G-Scale with SR algorithm with duration of request calculated as Exponential distribution with mean in $[2, 4, 6, 8, 10, 12]$. 

Figure 4.20: Average fraction guaranteed for type 2 and 3 flows in G-Scale with SRF algorithm with duration of request calculated as Exponential distribution with mean in $[2, 4, 6, 8, 10, 12]$. 

105
4.4.4 Discussion

As discussed in the previous section, the dynamic problem can be solved using greedy, greedy-fair, selective-rescheduling or selective-rescheduling-fair algorithms. The greedy algorithm was the basis of design of the other three algorithms greedy-fair, selective-rescheduling and selective-rescheduling-fair. A cloud provider must use the selective-rescheduling algorithm, if their aim is to maximize revenue due to hard deadline requests, or they can use greedy-fair algorithm if their objective is to maximize the network capacity allocated to non-deadline requests while minimally impacting the deadline requests and to maximize network utilization. It is to be noted, that greedy-fair algorithm generates lower acceptance rate for hard deadline requests as compared to selective-rescheduling as selective-rescheduling algorithm consistently prioritizes hard deadline requests over non-deadline requests. Selective-rescheduling-fair strikes a nice balance between the two algorithms by not trading off too much network capacity allocated to non-deadline requests to aggressively meet hard deadline requests. Hence, the provider must choose a solution and the corresponding trade-off based on their design requirement.

Amoeba [53] by Zhang et. al. provided hard deadline guarantees for only hard deadline requests, PGA [57] did not provide true guarantees at all and Tempus [29] only provided fairness among all flows. None of the related work provided guarantees for mix-flows simultaneously. Hence, we cannot provide an apple to apple comparison with these works.

4.5 Conclusion

In this chapter, we formulated the dynamic problem wherein our main objective was to find admission control, scheduling and routing decisions to provide guarantees for mix-flows under the assumption that the future transfer requests are not known to the system in advance. We proposed four algorithms, namely, greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair to solve the problem. We evaluated and compared the performance of the four algorithms to meet the guarantees. In the next chapter, we solve the dynamic problem in federated and multi-cloud environments.
In Chapters 3 and 4, we solved the static and the dynamic problem in a single cloud computing environment i.e., a single administrative domain owned by a single cloud provider. In this chapter, we extend the problem proposed in Chapter 4 by solving it in a federated-cloud computing environment, where two or more cloud administrative domains establish a common trust boundary and agree “out-of-band” how to mutually enforce policy and governance. We also solve the problem in a multi-cloud environment, where two or more cloud administrative domains do not have a common trust boundary and there is no agreement “out-of-band” how to mutually enforce policy and governance [9].

In Section 1.3, we discussed our research philosophy of adaptability. By solving the dynamic problem in a single cloud environment, we developed solutions that clearly modeled fairness vs. revenue trade-off for mix-flows, thereby providing adaptability in the face of changing customer demands and cloud provider goals. To provide adaptability in evolving cloud computing models, we solve the dynamic problem in federated-cloud and multi-cloud environments.

What makes solving the dynamic problem in federated-cloud and multi-cloud computing environments different from a single cloud computing environment is the fact that, instead of a single administrative domain, we now have multiple independent non coordinating administrative domains owned by different cloud providers with independent business objectives, which must depend on each other for their revenue generation.

In a federated-cloud, typically there is a third, independent entity, called the cloud exchange point, that provides connectivity and coordination among cloud providers, and depending on the policy used, anywhere from partial to complete network information may be shared with this entity. If we assume that the cloud exchange point has complete information about the
networks of the federating clouds, the problem becomes akin to that we already solved in Chapter 4 with the only difference being that the inter-datacenter network is a larger sized network comprising of two federating clouds. Such cloud computing systems are called open federated-cloud computing systems, since they assume complete transparency among resources being provisioned across clouds. However, since the cloud providers are competing entities with independent business objectives of maximizing revenue in some form, it is unrealistic to assume an open federated-cloud computing environment. Hence, hereon we assume only partial information is shared by the cloud providers with the cloud exchange points in a federated-cloud computing environment, and this is what makes the problem challenging. A cloud exchange point must allocate requests from one datacenter of a cloud provider to another datacenter of another cloud provider assuming incomplete end to end network information. In solving the dynamic problem in a single cloud environment, the controller running the solution algorithms had complete end to end inter-datacenter network information with respect to network paths and network utilization at each timeslot. Hence, we cannot directly apply the solutions proposed to solve the dynamic problem in a single cloud computing environment to a federated-cloud computing environment.

In a multi-cloud environment, typically there is no third party involvement, as there is no coordination among cloud providers, and no network information is shared among the providers. As a consequence, the problem of providing guarantees to mix-flows becomes all the more challenging. To provide guarantees, specifically hard guarantees, with incomplete network information then becomes a gamble. Doing that, would be an exercise in futility. Major cloud providers such as Google, Amazon and Azure, have full mesh connectivity with each other at the edge of their clouds [60]. With this assumption, we change the problem to finding the network capacity sufficient to meet the guarantees of mix-flows in inter-datacenter WAN environments.

In this chapter, we first discuss the design goals of Vritti in federated-cloud and multi-cloud. We then describe the system model in terms of traffic, network, revenue and fairness models. We then state the dynamic problem in federated-cloud and multi-cloud along with the corresponding control knobs. We then design algorithms to solve the dynamic problem. We end the chapter by evaluating the proposed algorithms.

## 5.1 Design Goals

In this section we describe the design goals of the proposed system - Vritti in a federated and multi-cloud computing environment. With these design goals we aim to manifest in Vritti our overarching research philosophy (see Section 1.3) of adaptability. Broadly, in Chapters 3 and 4 we provided adaptability by introducing a complete WAN traffic model with respect to deadline and volume characteristics and by providing a control knob to tune fairness vs. revenue
5.1.1 Meeting Cloud Providers’ Objectives in Federated-cloud

In a federated-cloud computing environment, the cloud providers involved establish a common trust boundary by sharing partial network information with a cloud exchange point as a conduit. But they may have independent business goals, because logically they are different administrative domains. Hence, it is only logical to assume that they would want to drive their system to meet their own business goals. Hence, with Vritti, our main objective is to allow federating cloud providers to achieve their own business objectives. We define the business objectives in terms of -

- Maximizing cloud provider revenue
- Providing fairness

A federating cloud provider should be able to meet one or both of these objectives. Do note, the two objectives are same as the first two design goals specified for dynamic problem in single cloud. To meet both the objectives together, a control knob will also be provided, as done for the previous problem.

What makes the problem tricky is that tenant requests now span more than one cloud provider. If we assume mix-flows like in the previous Chapters 3 and 4, we must be able to provide guarantees to all flow types. But the completion of a transfer is now dependent on more than one provider, with different network characteristics and different business objectives. The cloud exchange point as the independent entity providing connectivity and coordination among the cloud providers is also limited by the partial knowledge of the federating cloud networks. As a consequence, spatially and temporally allocating a request becomes all the more challenging because of the guarantees involved. For example, to guarantee a hard deadline request that generates in a datacenter DC1 in cloud provider CP1’s inter-datacenter network and is destined for a datacenter DC2 in cloud provider CP2’s inter-datacenter network, we must be able to find a schedule and a route within the inter-datacenter network of CP1 such that after traversing CP1’s network, CP2 has sufficient time and capacity to allocate the request before its hard deadline. However, it is not possible to predict the schedule and route for the inter-datacenter network of CP2 apriori with the greedy and greedy-fair algorithms, mainly because
we plan in the future one window at a time to reduce the problem size. Even if we do not plan one window at a time for hard deadline requests, as we do in the selective-rescheduling and selective-rescheduling-fair algorithms, we cannot know until CP1 has transferred the request, whether CP2 will be able to transfer the request or not, because we simply cannot skip ahead in time.

Also, maximizing revenue of a cloud provider as an objective should be taken with a grain of salt. Revenue due to an inter-cloud request is generated for all the cloud providers in the path of the request. As a consequence, the first inter-datacenter network that becomes a bottleneck in the path of the request, results in revenue loss for all the inter-datacenter cloud networks that had transferred the request before it reached the bottleneck network. So, while we can independently maximize the revenue of each cloud provider, we must account for the fact that by maximizing one cloud provider’s revenue we might adversely affect other cloud providers’ revenue. We take into account the tension between the different cloud providers’ in terms of meeting their business objectives while formulating the solution to the problem.

Another challenge in federated-cloud is providing strict guarantees to requests. Logically, strict guarantees can be provided to requests if we solve for the same LP specified in Section 3.3.2, as the guarantees are modeled as constraints. But now, since we have an additional bottleneck in terms of dependence on unknown networks for transfer completion, intuitively, the acceptance rate of requests with strict deadlines might drop. Hence, to achieve higher performance from the system, we may have to trade-off the strictness of the guarantees provided to tenant requests. We discuss this in more detail in in section 5.2.1.

5.1.2 Capacity Planning for Meeting Guarantees for Mix-flows in Multi-Cloud

In a multi-cloud environment, where there is no sharing of network information among cloud providers and no coordination among them, it becomes very difficult to elicit good performance from the algorithms while meeting all provider guarantees. To get good performance in the multi-cloud environment, we instead try to identify the network capacity needed.

The challenges in solving the dynamic problem in a federated-cloud environment also hold true for the multi-cloud environment. Rather, the challenges are amplified in the multi-cloud environment.

5.2 System Model

In this section we discuss the system model in terms of WAN traffic, network, revenue and fairness formulations.
5.2.1 WAN Traffic Model

The WAN traffic model remains the same as described in Section 3.2.1 for the static problem in a single cloud. The traffic specification and the flow types also remain the same.

However, as discussed in the previous section, due to an additional bottleneck introduced in the system by the virtue of it being a federated/multi cloud environment, the performance of the system may suffer. To avoid that, we must be able relax the traffic model and its corresponding guarantees. We discuss this in more detail from a cloud provider’s perspective in Section 5.3.1. By not considering all flow types in terms of deadline and volume, we cannot provide tailor-made guarantees for mix-flows. However, this dissertation is rooted with adaptability as the key objective. We can meet guarantees for all mix-flows as we see in Section 5.3.1, but from a provider point of view it would make little sense in providing strict guarantees, when resultant request acceptance rate is inadequate. Hence, we must trade-off completeness for higher performance.

5.2.2 Network Model - federated-cloud

The network model of a federated-cloud computing environment is a super-set of multiple single cloud computing environments. The single cloud computing environments remain the same as discussed in previous chapters. Each single cloud computing environment is an inter-datacenter network with a logically centralized SDN controller and each site equipped with a site broker. The SDN controller is responsible for generating scheduling and routing decisions for each request, based on which, a request is admitted or rejected. These decisions are enforced onto the underlying network via the site broker. Additionally, the SDN controller synchronizes with the site broker periodically, to update its network database. Each inter-datacenter network also has a set of datacenters acting as the cloud provider edge. In the federated-cloud these provider edges are connected to cloud exchange points, the entity that provides connectivity among multiple cloud provider domains. In the multi-cloud environment these provider edges are connected in a full-mesh network [60]. We illustrate the system architecture of the federated-cloud and multi-cloud computing environments in Figures 5.1 and 5.2, respectively. As can be seen in Figure 5.1, the cloud exchange point works at both the control plane level and the data plane level. The SDN controllers communicate with each other via the cloud exchange points along the control plane. A request is transferred from a provider edge in a cloud provider to another provider edge in another cloud provider via a cloud exchange point along the data plane.
Figure 5.1: System Architecture for Federated-Cloud.

Figure 5.2: System Architecture for Multi-Cloud.
**Assumptions**

- We assume, multiple clouds in federated environment are connected using software-controlled cloud exchange points, which have the following features (we make these assumptions on the basis of work done by Rao [59] and Demchenko [24] in the field of federated-cloud computing):

  - The cloud exchange points are partially open exchange points, in the sense they have partial network information i.e., only the network topology connecting the exchange point to the datacenters which are participating in the communication. However, they are neutral entities. If cloud exchange points were fully open, the problem would have transformed into a dynamic problem in a single cloud environment just with a bigger underlying network, making the problem trivial. On the other hand, closed exchange points would make the system akin to a multi-cloud environment. Hence, we assume partially open exchange points.

  - They are software-controlled, in the sense, they can run algorithms such as those used by Vritti. This assumption is made for simplicity.

  - These cloud exchange points have a software interface to accept service requests that are required to be serviced by more than one cloud provider.

  - They also control a portion of the edge network; determining one or more particular, physical devices of the edge network that are usable to serve the request, and configuring physical devices of the edge network to provide the service [59]. This assumption is made to define the flow of requests along the data plane.

  - SDN controllers for each cloud provider are connected directly to the cloud exchange points. This assumption is made for easier control plane communication.

- For simplicity, in here we do not allow one or many cloud provider networks to be used as transit for the source and destination clouds. *There exist only two cloud providers in federation with each other in a federated-cloud environment, two cloud providers peering with each other in a multi-cloud environment.* This is an important assumption. Considering the complexity of the problem, we chose to solve the problem in a base federated and multi-cloud computing environment with two administrative domains. We discuss in future work, how we intend to extend the solution to a federated-cloud with more than two providers.

- In multi-cloud environment, we assume direct peering among the provider edges of the cloud.
5.2.3 Revenue and Fairness Model

The revenue and fairness model for each of the cloud provider remains the same as discussed in Chapters 3 and 4. Essentially, we extend the dynamic problem in a single cloud to a federated and multi-cloud environment. Hence, the revenue and fairness definitions for the federating and multi-cloud computing environments remain the same.

5.3 Dynamic Problem

In this section we state the exact problem statements and identify the main control knobs.

5.3.1 Federated-cloud: Exact Problem Statements

We formulate the problem statements from both a tenant’s perspective and from a cloud provider’s perspective. From a tenant’s point of view, the objective is to get guarantees for mix-flows. As a tenant, we don’t care about the underlying objective of the cloud provider. From a cloud provider’s perspective, the objective is to maximize revenue in terms of request acceptance rate. We state the exact problem statements below -

Problem 1 - Tenant’s Perspective

- **Given** two cloud providers in federation with each other via a cloud exchange point.
- **Given** also transfer requests by each tenant, which are generated by a datacenter of one cloud provider and destined for a datacenter of the other cloud provider, and vice versa, in federation with each other.
- The **objective** is to identify admission control, scheduling and routing decisions for the mix-flow of requests such that revenue for each provider is greedily maximized from cloud provider’s perspective and all type 1-5 tenant requests are provided their respective guarantees.

Problem 2 - Cloud Provider’s Perspective

- **Given** two cloud providers in federation with each other via a cloud exchange point.
- **Given** also transfer requests by each tenant, which are generated by a datacenter of one cloud provider and destined for a datacenter of the other cloud provider, and vice versa, in federation with each other.
- The **objective** is to identify the flow guarantees which when provided, the revenue for each provider is greedily maximized from the cloud provider’s perspective.
5.3.2 Multi Cloud: Exact Problem Statement

Given two cloud providers in peering with each other directly.

Given also transfer requests by each tenant, which are generated by a datacenter of one cloud provider and destined for a datacenter of the other cloud provider, and vice versa, peering with each other.

The objective is to identify the network capacity sufficient to meet the flow guarantees such that the revenue for each provider is greedily maximized from the cloud provider’s perspective.

5.3.3 Control Knobs

The main traffic engineering control knobs remain the same i.e., admission control, scheduling and routing. Every SDN controller of a cloud provider generates scheduling and routing decisions. If the destination cloud providers is able to meet the request guarantee, the request is admitted, else rejected. The main control knobs remain the same, as the underlying decisions for a request per cloud provider remain the same.

However, now as the request is to be served by both the federating cloud providers, we divide the end time or deadline of the request such that both cloud providers get sufficient time to complete the request. The ratio in which the request deadline or request end time is divided is an additional control knob used. As soon as a request arrives, the ratio is calculated, and deadline for the request at the source inter-datacenter network, and the start time for the request at destination inter-datacenter network is updated.

Another control knob is the selection of the provider edge at both the cloud providers to transfer the request. For the purpose of this dissertation we fix the provider edges for simplicity.

5.4 Algorithm Design

In this section we design the algorithms used to solve the dynamic problem in federated and multi-cloud environments.

The idea is to solve the problem for obtaining scheduling and routing decisions at each cloud provider separately, while the admission control decision is generated at the inter-mediating cloud exchange point. To reduce the problem size for each cloud provider, we use the same method as used for a single cloud provider i.e., we divide time into timeslots, and k timeslots constitute a time window. The planning for the future is done at a window level granularity for all flow types in the greedy and greedy-fair algorithms. In selective-rescheduling and selective-rescheduling-fair, we plan for a window at a time for soft and non-deadline requests, and plan for the entire duration of the request for hard deadline requests.
We extend the greedy, selective-rescheduling, greedy-fair and selective-rescheduling-fair algorithms proposed in Chapter 4 such that they are applicable to the federated-cloud environment. We extend the greedy algorithm s.t. it is also applicable to multi-cloud environment.

5.4.1 Greedy Algorithm

In this section we describe the greedy algorithm in terms of its objectives and formulate the greedy solution to the dynamic problem in Algorithms 5 and 6 for federated-cloud and multi-cloud, respectively.

Algorithm Description

With the greedy algorithm our main objective is to provide a straight-forward approach to solving the dynamic problem to allow computational feasibility while meeting mix-flow guarantees and greedily maximizing revenue for both the cloud providers.

The algorithm allocates a request in space and time such that its guarantees can be met and both providers’ revenue is maximized. Any request which cannot be provided its required guarantee is rejected for good, unless the tenant resubmits the request. In the federated-cloud environment, for all requests that arrive in a timeslot destined to an external cloud provider, the requests are divided on the basis of their deadlines among the two cloud providers. In multi-cloud, we do not divide requests among the different cloud providers, because of the lack of coordination among the providers. For every inter-cloud request, a datacenter in the inter-datacenter network of the source cloud provider is selected as a destination for the request, and its deadline is also updated. Also, a datacenter in the inter-datacenter network of the destination cloud provider is selected as a source for the request at the destination cloud provider, and its start time is updated according to the deadline selected at the source cloud provider. Then we run steps 2 to 5 of the Algorithm 1 proposed in Chapter 4 for the source cloud provider. If the source cloud provider is able to provide the guarantee associated with the request type, the request for the destination cloud provider is updated accordingly. If the source cloud provider is not able to meet the guarantee associated with the request, the corresponding request that was scheduled to arrive at the destination cloud provider is deleted. Otherwise, we again run steps 2 to 5 of the Algorithm 1 proposed in Chapter 4 but this time for the destination cloud provider. If the destination cloud provider is also able to meet the guarantee associated with the request, request is accepted. For each federated-cloud request accepted or rejected, we update the factor that divides the request along time for the two cloud providers. It is with this factor $\tau$ we decide how much time must be allocated to a cloud provider based on the time taken by the provider, historically, to allocate requests of similar throughput requirement. Algorithm 5 describes the greedy algorithm in federated-cloud. Algorithm 6 describes the greedy algorithm
To make the problem computationally feasible, we reduce the problem size by planning for a window at a time as done for greedy algorithm in single cloud environment.

In terms of providing guarantees to mix-flows, we must note that the guarantees for deadline requests are formulated as constraints in the LP formulation. Hence, if an LP is able to find a feasible solution in a timeslot, it means it is able to meet all the constraints and also the guarantees. Else, the hard deadline requests are rejected, while other requests are accepted as they do not have hard guarantees.

With this greedy approach we greedily try to maximize revenue due to all flow types by allocating partial requests between the current timeslot and end time of the current window.

**Algorithm 5** Greedy Algorithm in Federated-cloud.

| Input: | Network Topology: $G(\mathcal{N}, \mathcal{E})$, $y_k(t, i) = \{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d1,t,i}, k_{d2,t,i}, V_{t,i}, P(t, i)\}$, where $y$ is a request generated in timeslot $k$. |
| | $Y_{k, \text{all}} = \{y_{1,1}, y_{1,2}, ..., y_{t,i} \}$, all tenant requests generated in timeslot $k$. |
| PE dict | $\{CP_j : \{PE_{1,j}, PE_{2,j}, ..., PE_{l,j}\}\}$ where $CP_j$ is a $j^{th}$ cloud provider peered with the source cloud and has $l$ provider edges peered with it. |
| CP | $CP_i$ is the cloud provider with which requests are generated. |
| Output: | Admission, Scheduling and Routing |
| 1: | for Requests arrived in Time Slot $k$ of Time Window $w$ do |
| 2: | if Request is a federated-cloud request with external cloud destination such as $CP_j$ then |
| 3: | Update its deadline by a factor $1/\tau$ |
| 4: | Select the PE from the of $PE_{\text{dict}}$ based on lowest average path cost |
| 5: | Update the destination of the request to the PE selected |
| 6: | Replace the federated-cloud request in $Y_{k, \text{all}}$ with the updated request |
| 7: | Run steps 2 - 5 of the Algorithm 1 |
| 8: | for Each federated-cloud request allocated by Greedy do |
| 9: | Send request to $CP_j$ with updated start time and original deadline |
| 10: | Poll $CP_j$ to get status of the federated-cloud requests |
| 11: | for Each federated-cloud request accepted or rejected by $CP_j$ do |
| 12: | Update $\tau$ based on federated-cloud throughput map |

### 5.4.2 Selective Rescheduling Algorithm

The selective-rescheduling algorithm, as in the case of single cloud, selectively reschedules already rejected requests to improve overall request acceptance rate. In this section we describe the approach and formulate the algorithm.
Algorithm 6 Greedy Algorithm in Multi-cloud.

Input: Network Topology: \( G(N,E) \), \( y_k(t,i) = \{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_1,t,i}, k_{d_2,t,i}, V_{t,i}, P(t,i), CP_j \} \). where \( y \) is a request generated in timeslot \( k \).

\( Y_{k,all} = \{y_{1,1}, y_{1,2}, ..., y_{1,i}, ..., y_{t,i}\} \), all tenant requests generated in timeslot \( k \).

\( PE_{dict} = \{CP_j : [PE_{1,j}, PE_{2,j}, ..., PE_{l,j}]\} \) where \( CP_j \) is a \( j^{th} \) cloud provider peered with the source cloud and has \( l \) provider edges peered with it.

\( CP_i \) is the cloud provider with which requests are generated.

Output: Admission, Scheduling and Routing

1: for Requests arrived in Time Slot \( k \) of Time Window \( w \) do
2: if Request is a multi-cloud request with external cloud destination such as \( CP_j \) then
3: Select the PE from the of \( PE_{dict} \) based on lowest average path cost
4: Update the destination of the request to the PE selected
5: Replace the multi-cloud request in \( Y_{k,all} \) with the updated request
6: Run steps 2 - 5 of the Algorithm 1
7: for Each multi-cloud request allocated by Greedy do
8: Send request to \( CP_j \) with updated start time and original deadline
9: Poll \( CP_j \) to get status of the multi-cloud requests
10: for Each multi-cloud request accepted or rejected by \( CP_j \) do

Algorithm Description

The idea behind this approach is to accommodate already rejected requests. As in the case of selective-rescheduling in single cloud, we cannot claim certainly that the rescheduled request will be allocated, as we cannot predict future requests in a realistic cloud computing environment, and as a consequence, we do not have full control over the input being generated for the LP in a timeslot. Also, we use a sliding window approach unlike the greedy algorithm, and plan the schedule and route for hard deadline requests for their entire duration to provide true apriori guarantees.

Similar to the greedy algorithm in federated-cloud, we first divide the request among the two cloud providers along time, and then run steps 2 to 12 of Algorithm 3 for each of the two cloud providers. The working of the selective-rescheduling remains the same as that for the single cloud. The only change is the dynamic division of the request among the two cloud providers to meet the guarantees of mix-flows and maximize their revenue. Algorithm 7 describes the complete algorithm.

To make the problem computationally feasible, we split all the non-hard deadline requests and non-deadline requests, hence, reducing the problem size.

We clearly state that with selective-rescheduling we cannot guarantee a rescheduled rejected request will be allocated. This is attributed to the randomness in the input requests generated at every timeslot. However, when the algorithm is run for sufficiently long time, we believe,
again attributing to unpredictable future requests, that by rescheduling rejected requests, we increase the odds of the request being allocated in a future timeslot.

With this algorithm, we trade off additional complexity and computation time for a higher revenue. Moreover, we trade off bandwidth allocation to non-deadline and soft deadline flows for higher acceptance rate.

**Algorithm 7** Selective Rescheduling Algorithm in Federated-cloud.

**Input:** Network Topology: $G(N,E)$, $y_k(t,i) = \{S_{t,i}, D_{t,i}, k_{s,t,i}, k_{d_{1,t,i}}, k_{d_{2,t,i}}, V_{t,i}, P(t,i), CP_j\}$.

where $y$ is a request generated in timeslot $k$,

$Y_{k,all} = \{y_{1,1}, y_{1,2}, ..., y_{1,i}, ..., y_{t,i}\}$, all tenant requests generated in timeslot $k$.

$PE_{dict} = \{CP_j : \{PE_{E_{1,j}}, PE_{E_{2,j}}..PE_{E_{l,j}}\}\}$ where $CP_j$ is a $j^{th}$ cloud provider peered with the source cloud and has $l$ provider edges peered with it.

$CP_i$ is the cloud provider with which requests are generated.

**Output:** Admission, Scheduling and Routing

1: **for** Requests arrived in Time Slot $k$ of Time Window $w$ **do**
2: **if** Request is a federated-cloud request with external cloud destination such as $CP_j$ **then**
3: Update its deadline by a factor $1/\tau$
4: Select the PE from the of $PE_{dict}$ based on lowest average path cost
5: Update the destination of the request to the PE selected
6: Replace the federated-cloud request in $Y_{k,all}$ with the updated request
7: Run steps 2 - 12 of the Algorithm 3
8: **for** Each federated-cloud request allocated by Greedy **do**
9: Send request to $CP_j$ with updated start time and original deadline
10: Poll $CP_j$ to get status of the federated-cloud requests
11: **for** Each federated-cloud request accepted or rejected by $CP_j$ **do**
12: Update $\tau$ based on federated-cloud throughput map

**5.4.3 Greedy and Selective Rescheduling Algorithms with Fairness Objectives**

In this section we extend the greedy and selective-rescheduling algorithms by adding a fairness component to both the algorithms.

**Algorithm Description**

With both the greedy-fair and selective-rescheduling algorithms, we intend to augment the algorithms by adding a fairness component. As in the case of greedy-fair and selective-rescheduling-fair in single cloud, with the additional fairness component, we intend to improve the bandwidth
allocated to non-deadline requests by minimally trading off bandwidth allocated to deadline requests. In the federated-cloud scenario, we divide the requests among the two cloud providers in an adaptive fashion, wherein the time allocated to a cloud provider to meet the request changes based on the historic performance of the cloud provider in allocating the requests of similar throughput requirement.

In the greedy-fair algorithm, similar to the greedy algorithm in federated-cloud, we first divide the request among the two cloud providers along time, and then run steps 2 to 10 of Algorithm 2 for each of the two cloud providers. The core functionality of the greedy-fair remains the same as that for the single cloud. The only change is the dynamic division of the request among the two cloud providers to meet the guarantees of mix-flows and maximize their revenue.

Similarly, in the selective-rescheduling-fair algorithm, we first divide the request among the two cloud providers along time, and then run steps 2 to 17 of Algorithm 4 for each of the two cloud providers, with the core functionality remaining the same as that of the algorithm in single cloud.

To make the problem computationally feasible, we split all the non-hard deadline requests and non-deadline requests in selective-rescheduling-fair and split all the requests in greedy-fair, hence, reducing the problem size.

Similar to greedy-fair and selective-rescheduling-fair algorithms in single cloud, we are utilizing the network capacity which would have been otherwise wasted to allocated capacity to non-deadline requests explicitly. Hence by minimally trading off bandwidth allocation to deadline requests, we are providing fairness to non-deadline requests. However, this come with an additional cost of computational time, as we solve more linear programs as compared to greedy and selective-rescheduling algorithms.

In Figure 5.3, we summarize how the algorithms can be modified to adapt in a federated-cloud environment.
5.5 Simulation and Evaluation

In this section we evaluate the performance of the three algorithms proposed to solve the dynamic problem.

5.5.1 Simulation Environment and Performance Metrics

The simulation environment is same as that of the static and the dynamic problem in single cloud (see Section 3.4.1). We run the simulation for 30 timeslots, 30 times. We chose 30 timeslots, mainly due to high compute capacity required. We ran the experiments on the compute available through NC State University’s High-Performance Computing (HPC) Services [67].

The performance metrics remain the same as that of the static and dynamic problems in single cloud, namely (see Section 3.4.1), average request acceptance rate, average fraction guaranteed, average fraction of type 4 requests allocated, average fraction of type 5 requests allocated, average network utilization, average throughput, average revenue, time taken, CPU usage and memory usage.
5.5.2 Top Level Questions

1. **How effective is the algorithm in providing guarantees to mix-flows to both cloud providers?** - With this question, we aim to analyze the effectiveness of the three algorithms in providing guarantees to mix-flows
   
   - Type 1 and 3 requests - request acceptance rate?
   - Type 2 and 3 requests - average fraction guaranteed?
   - Type 4 and 5 requests - average fraction allocated?

2. **How does network utilization vary with request arrival rate for both cloud providers?** - With this question, we aim to evaluate the algorithms with respect to average network utilization in lightly loaded and highly loaded conditions.

3. **How does throughput vary with request arrival rate for both cloud providers?** - With this question, we aim to evaluate the algorithms with respect to average throughput in lightly loaded and highly loaded conditions.

4. **How does space and time overhead of the algorithms vary with request arrival rate?** - With this question, we aim to evaluate the performance of the algorithms w.r.t. allocation time and memory usage.

5. **How effective is the algorithm in providing guarantees to only hard deadline requests for both cloud providers?** - With this question, we aim to identify the performance of the algorithms with only type 1 requests.

6. **How effective is the algorithm in providing guarantees to only soft deadline requests for both cloud providers?** - With this question, we aim to identify the performance of the algorithms with only type 2 requests.

7. **How effective is the algorithm in providing guarantees to only soft-hard deadline requests for both cloud providers?** - With this question, we aim to identify the performance of the algorithms with only type 3 requests.

8. **How effective is the algorithm when allocating only non-deadline requests for both cloud providers?** - With this question, we aim to identify the performance of the algorithms with only type 4 and type 5 requests.

9. **How much network capacity must be allocated in a multi-cloud environment to get similar performance as that in single cloud?** - With this question, we intend to identify a sufficient range for capacity planning in multi-cloud environments given the mix-flow input.
5.5.3 Simulation Results

In this section, we evaluate the results of the simulation to answer the top level questions proposed.

Q1. How effective is the algorithm in providing guarantees to mix-flows to both cloud providers?

- **Type 1 and 3 requests with Greedy, GF, SR and SRF** - We evaluate the allocation of hard deadline requests i.e., type 1 and type 3 requests with percentage request acceptance rate w.r.t. arrival rate. A high acceptance rate signifies a larger percentage of hard deadline requests were fully completed by the hard deadline, and a lower acceptance rate signifies the contrary. As we can see in Figure 5.4, the acceptance rate for both cloud providers is similar but not exactly the same, this is because acceptance rate of the cloud providers are calculated on the basis of their input requests, and not the global request input. This means if a cloud provider generating a request is not able to allocate it, the request will not be accounted for in calculating acceptance rate of the destination cloud provider. As in the case of single cloud environment, SR shows higher acceptance rate as compared to the SRF, greedy and GF algorithms. However, a notable difference is the drastic drops in the acceptance rate due to both SR and greedy in federated-cloud environment as compared to single cloud environment. This is mainly because of the inherent interdependence between cloud providers, which becomes a major bottleneck when meeting requests with strict deadline requirements.
• Type 2 and 3 requests with Greedy, GF, SR and SRF - We evaluate the allocation of soft deadline requests i.e., type 2 and type 3 requests with average fraction guaranteed w.r.t. arrival rate. A high average fraction guaranteed signifies, on an average for soft deadline requests a larger fraction of demand requested was completed by the soft deadline, and a lower average fraction guaranteed signifies the contrary. As we can see in Figure 5.5, the average fraction guaranteed decreases with increasing arrival rate. Like in the single cloud environment SR and SRF observe a lower average fraction guaranteed as compared to greedy and GF algorithms. This is due to the explicit highest priority given to hard deadline requests by SR and SRF, in terms of both long term reservation and rescheduling of requests. Average fraction guaranteed for both the cloud providers is similar for both the algorithms, as in the case of acceptance rate.
Figure 5.5: Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale.

- **Type 4 and 5 requests with Greedy, GF, SR and SRF** - We evaluate the allocation of non-deadline requests i.e., type 4 and type 5 requests with average fraction allocated w.r.t. arrival rate. A high average fraction allocated signifies, on an average for non-deadline requests, a larger fraction of demand requested was completed, and a lower average fraction allocated signifies the contrary. As we can see in Figures 5.6 and 5.7, the average fraction allocated decreases with increasing arrival rate for greedy, SR and SRF. SR and greedy algorithm perform the worst because they are not designed to provide explicit fairness to non-deadline requests; rather, they are designed to accommodate more hard deadline requests. Hence, they trade-off bandwidth allocated to non-deadline requests to meet more hard deadline requests.
Q2. How does network utilization vary with request arrival rate for both cloud providers?

Network utilization increases with request arrival rate as can be seen in Figure 5.8. This is expected with an increase in arrival rate. GF outperforms the remaining three algorithms with
network utilization as high as 58% at arrival rate of seven. This is because, greedy-fair utilizes the capacity of the network whenever the LP becomes infeasible, by updating the LPs with relaxed constraints, and allocating non-deadline requests along space and time, which would have not been utilized otherwise. Greedy-fair algorithm again clearly outperforms SR and greedy when it comes to effectively utilizing the network by accommodating non-deadline requests with little drop in acceptance rate for hard deadline requests.

Figure 5.8: Average Network Utilization in G-Scale.

Q3. How does throughput per job vary with request arrival rate for both cloud providers?

Average throughput per job calculated in Gb per timeslot decreases with request arrival rate as can be seen in Figure 5.9. While greedy algorithm has consistently lower throughput as compared to GF, SR and SRF, GF achieves higher average throughput per job for higher request arrival rate as compared to SR. This is mainly because, the non-deadline requests in greedy-fair algorithm contribute higher throughput as compared to hard deadline requests in SR. The mirroring per job throughput for the two cloud providers with GF is due to the interdependency among the two cloud providers in meeting requests. Since, we try to greedily meet each provider’s objectives, a decrease in throughput provided by one cloud provider will come at the cost of higher throughput provided by the other cloud provider.
Q4. How does space and time overhead of the algorithms vary with request arrival rate?

As can be seen in Figure 5.10, time taken to generate spatial temporal allocation of requests is the highest for SR and SRF because they are designed to give highest priority to hard deadline requests. As can be seen in Figure 5.11, memory usage by SR and GF is lower as compared to greedy and SRF algorithms.

Figure 5.9: Average Throughput in G-Scale.

Figure 5.10: Time Taken in G-Scale.
Q5. How effective is the algorithm in providing guarantees to only hard deadline requests for both cloud providers?

To answer this question we evaluate the performance of greedy algorithm with only hard deadline requests as input. As we can see in Figure 5.12, the acceptance rate of hard deadline requests decreases with time. Even though there are no other request types, the acceptance rate of type 1 requests is very low. This is mainly because of the all or nothing approach used to allocate the requests.
Q6. How effective is the algorithm in providing guarantees to only soft deadline requests for both cloud providers?

To answer this question we evaluate the performance of greedy algorithm with only soft deadline requests as input i.e., type 2 requests. As we can see in Figure 5.13, the average fraction guaranteed for soft deadline requests decreases with time. The average fraction guaranteed to type 2 requests is lower than when all request types are present, however, it is still much higher than the acceptance rate of type 1 requests. Hence, if we are to select a guarantee to provide to request types in federated environment, this is a better guarantee than the all or nothing guarantee provided to hard deadline requests.
Q7. How effective is the algorithm in providing guarantees to only soft-hard deadline requests for both cloud providers?

To answer this question we evaluate the performance of greedy algorithm with only soft-hard deadline requests as input i.e., type 3 requests. As we can see in Figure 5.15, the average fraction guaranteed for soft-hard deadline requests decreases with time, and in Figure 5.14, the acceptance rate for soft-hard deadline request also decreases with time. Both the acceptance rate and average fraction guaranteed for type 3 requests are low, acceptance rate for these requests is same as type 1 request i.e., 20% at the lowest arrival rate of 1 and declines to zero as arrival rate increases. Average fraction guaranteed is more realistic as a guarantee as we can see in Figures.
Figure 5.14: Acceptance Rate for Type 3 Flows in G-Scale if Only Type 3 Requests in Workload.

Figure 5.15: Average Fraction Guaranteed for Type 3 Flows in G-Scale if Only Type 3 Requests in Workload.
Q8. How effective is the algorithm when allocating only non-deadline requests for both cloud providers?

To answer this question, we evaluate the performance of greedy algorithm with only non-deadline requests as input i.e., type 4 and type 5 requests. As we can see in Figures 5.16 and 5.17, the average fraction allocated for non-deadline requests decreases with time. However, non-deadline requests are not provided a hard guarantee, but a fairness objective in contrast to deadline requests. That is the reason the average fraction allocated to type 4 and type 5 request is much higher as compared to deadline requests.

![Figure 5.16](image)

**Figure 5.16:** Average Fraction Allocated for Type 4 Flows in G-Scale if Only Type 4 Requests in Workload.
Q9. How much network capacity must be allocated in a multi-cloud environment to get similar performance as that in single cloud?

To answer this question we evaluate the performance of greedy algorithm in the multi-cloud environment with mix-flows with network capacity ranging from 16 Gbps to 160 Gbps. As we can see in Figures 5.18, 5.19, 5.20 and 5.21, the higher the network capacity, the higher the performance. There is direct correlation between the network capacity, and the network capacity allocated to meet the request.
Figure 5.18: Acceptance Rate for Type 1 and Type 3 Flows in G-Scale.

Figure 5.19: Average Fraction Guaranteed for Type 2 and 3 Flows in G-Scale.
5.5.4 Discussion

Through questions one to four, we answered the first part of the dynamic problem, i.e., meeting guarantees for mix-flows in a dynamic federated-cloud environment. Through questions five
to eight, we answered the second problem, i.e., identifying which guarantee must be provided to the traffic in federated-cloud environments. SRF algorithm performs better as compared to greedy, GF and SR algorithms in terms of providing higher acceptance rate to hard deadline requests, by minimally trading off capacity from non-deadline requests. In terms of identifying what kind of guarantees must be provided in the federated-cloud environment, we can clearly state that it is the costliest to provide hard deadline guarantees when dealing with multiple administrative domains. The safest option is to provide soft deadline guarantees and fairness to non-deadline requests. For the multi-cloud, we were able to identify that with ten times increase in network capacity there is proportional increase in the performance of the greedy algorithm in terms of all the guarantees provided.

5.6 Conclusion

In this chapter, we proposed the dynamic problem wherein our main objective was to find admission control, scheduling and routing decisions to provide guarantees for mix-flows, under the assumption that the future transfer requests are not known to the system in advance in the federated-cloud and multi-cloud environments. We modified four algorithms, namely, greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair to solve the problem in federated-cloud. We evaluated and compared their performance and ability to meet the guarantees. We also solved the dynamic problem in multi-cloud environment for the capacity planning question using modified greedy algorithm.
6.1 Conclusion

In this dissertation, we presented Vritti, a system that provides guarantees for mix-flows in inter-datacenter wide area networks in three system environments, namely, single cloud, federated-cloud and multi-cloud. For the single cloud environment, we formulated the static problem (wherein the future tenant demands are known to the system in advance) as a linear program with guarantees modeled as constraints. We then proposed a more realistic dynamic problem (wherein the future tenant demands are not known to the system in advance). We proposed four algorithms to solve the dynamic problem, namely, greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair. We did extensive evaluation of the algorithms using simulations. We solved the dynamic problem in federated-cloud environment by modifying the greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair algorithms. For simplicity, we solved the dynamic problem in multi-cloud environment by modifying the greedy algorithm. We summarize the problems solved in the following section in more detail.

6.1.1 Problems Solved

In this section, we revisit the three problems we proposed to solve in Chapter 1. We discuss how we solved the problems.

Q1. How to provide tailor-made guarantees for mix-flows in inter-datacenter wide area networks?

To the best of our knowledge, no known work handles deadline and non-deadline flows in conjunction in an inter-datacenter wide area network environment. To encompass the different deadline and volume requirements of traffic seen in most cloud computing environments, we
classified traffic into five flow types, which we call mix-flows. We provide tailor-made guarantees to these mix of flows. For type 1 flows or hard deadline requests, we guarantee full transfer completion by the hard deadline. For type 2 flows or soft deadline requests, we guarantee partial transfer completion before the soft deadline. For type 3 flows or soft and hard deadline flows, we guarantee partial completion by the soft deadline, and full transfer completion by the hard deadline. For type 4 and 5 flows i.e., non-deadline flows with known volume and non-deadline flows with unknown volume, respectively, we provide fairness. We formulate the problem of identifying spatial temporal allocation decisions for mix-flows as a linear program and the guarantees for these flows as constraints to the linear program. We provide guarantees for type 1, 2, 3, 4 and 5 flows using the greedy, greedy-fair, selective-rescheduling and selective-rescheduling-fair algorithms. At the heart of the four proposed algorithms, we maximize revenue due to all flow types, where we formulate revenue due to a request mathematically as a function of requested throughput and average path cost for the request. A request which has a higher throughput requirement and has higher average path cost, generates higher revenue, the request can be a deadline request or non-deadline request. We prioritize deadline requests over non-deadline requests by using all-or-nothing guarantee constraints for the former. With the greedy-fair and selective-rescheduling-fair approaches, we provide fairness to non-deadline flows by allocating them in lieu of deadline flows periodically.

Q2. How to achieve revenue versus fairness trade-off while providing guarantees to mix-flows?

We realized it is very difficult to heuristically trade-off revenue due to deadline requests for non-deadline requests, using a single linear program and an all-or-nothing approach. This is mainly because of the strictness of equivalence constraints for deadline requests. Hence, in the greedy-fair and selective-rescheduling-fair algorithms we modify the linear program by relaxing its constraints periodically to meet the non-deadline requests in lieu of deadline requests. The number of times the updated linear program is employed to prioritize non-deadline requests over deadline requests is the control knob in the hand of the cloud provider to allow feasible revenue vs. fairness trade-off.

Q3. How to modify a system designed to provide transfer guarantees to fit into federated and multi-cloud environments?

We also solve the same dynamic problem in federated and multi-cloud environments. We assume the base case of two administrative domains i.e., two cloud providers, due to computational constraints. The complexity of the problem in these system environments varies depending on the level of information exchange allowed among the multiple administrative domains. In the
federated-cloud environment, we assume partial exchange of information among cloud providers, and in multi-cloud environment, no exchange of information. We modify greedy, greedy-fair, selective rescheduling and selective-rescheduling-fair algorithms to meet the guarantees for mix-flows in federated-cloud environment. For simplicity, we modify only greedy algorithm to meet the guarantees for mix-flows in multi-cloud environment. For the federated-cloud, since we assume some cooperation between two clouds, provided by a cloud exchange point, we dynamically divide the requests along time among the two cloud providers. The inter-dependence among the two cloud providers turns out to be the main bottle-neck when allocating requests, the largest brunt of which is paid by hard deadline requests because of their strict guarantee constraints. Soft deadline and non-deadline requests which have more relaxed guarantee constraints see a relatively higher fraction of requests being allocated. We solve the dynamic problem in federated-cloud in two scenarios, first, with all mix-flow requests as input and second with only one request type as input. In the first scenario, our objective was to evaluate the performance of the modified algorithms in federated-cloud environment. In the second scenario, our objective was to identify the type of guarantees that must be provided to traffic traversing the federated-clouds to achieve higher request allocation rate. Guarantees provided to soft deadline and non-deadline request resulted in higher request allocation rate in the second scenario.

6.2 Future Work

We propose the following research directions for the future.

6.2.1 Theoretical Analysis of the Algorithms

In this dissertation, we evaluate the performance of the algorithms using simulations. To provide a stronger case of performance guarantee, the logical next step is to supplement that with a theoretical performance guarantee by calculating the competitive ratio of the proposed online (dynamic) algorithms with respect to the offline (static) optimal algorithm. Competitive analysis is commonly used to evaluate online algorithms, where the performance of an online algorithm is compared to that of the optimal offline algorithm [1]. If the ratio of the performance of an online algorithm over that of the optimal offline algorithm is bounded, then the online algorithm is considered to be competitive and that ratio is called a competitive ratio [57].

6.2.2 TCP for Mix-flows in Inter-DC Networks

Vritti provides guarantees for mix-flows in inter-datacenter wide area network environment, wherein we guarantee deadlines for deadline flows and provide fairness to non-deadline flows. We assume multi-path routing to meet Vritti’s objectives. It would be interesting to explore the
inter-play between TCP and Vritti. In some prior work [4], [20], [21] TCP has been modified and new transport protocols have been proposed to support deadlines and multi-path routing in intra-DC networks. It has also been shown that a substantial fraction (from 7% to over 25%) of flow deadlines are not met using TCP in a study of multiple production DCNs [12]. Chen et al. [39] proposed a minimal impact congestion control protocol to handle deadline flows with as little bandwidth as possible. But there has been no work that has studied the impact of TCP in providing guarantees to mix-flows in inter-datacenter network environment. The impact of Vritti on TCP and its modifications such as DCTCP [4], D2TCP [20], MPTCP [21] and MCP [39] is an interesting topic for future research. In this dissertation we study the problem of scheduling and routing of mix-flows in inter-datacenter networks at flow level granularity. Another valuable research direction is studying the impact of packet reordering and guaranteeing deadlines at flowlet and co-flow level granularity.

6.2.3 Large Scale Implementation of Algorithms for Federated and Multi-Cloud Environments

In this dissertation, we implement solutions for federated and multi-cloud environment assuming two administrative domains or cloud providers. However, in a realistic environment two administrative domains can be connected by one or more different administrative domains. This significantly increases the scale of the problem, because of the number of cloud providers’ networks involved. Moreover, this also significantly increases the complexity of the problem, because of the impact on cloud provider revenue due to interdependence among them, when transferring requests. Vritti sees a significant drop in performance when moving from single cloud to federated-cloud to multi-cloud environments when providing guarantees for mix-flows, even under the base case of two cloud providers. It will be challenging to design algorithms that improve the performance of Vritti in federated-cloud and multi-cloud environments, and extending those algorithms such that the performance does not deteriorate substantially with scale.

6.2.4 Point to Multi-point Resource Allocation in Inter-DC Networks

In this dissertation we developed algorithms for fast and efficient point to point transfers, and evaluated them using simulations. Extending the techniques proposed for point to multi-point transfers in inter-datacenter networks with the objective of providing guarantees for mix-flows is yet another research direction to explore. These are requests that transfer demands from one datacenter to multiple datacenters. For example, content delivery networks (CDNs) may push video content to regional cache locations [36, 14, 13, 35], cloud storage services may replicate data objects across multiple sites for increased reliability [38, 16], and search engines may push
substantial updates to their geographically distributed search database on a regular basis [26]. Data transfers among datacenters for replication of objects from one datacenter to multiple datacenters is referred to as geo-replication [26, 28, 3, 14, 42, 40, 51] and can form a large portion of inter-datacenter traffic [47]. Studying workload comprising of both P2P and P2MP transfers in conjunction is also an interesting future research direction.
Bibliography


Yuan Feng, Baochun Li, and Bo Li. “Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward”. In: Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on. IEEE. 2012, pp. 43–50.


[65] IBM Cloud. IBM Cloud.

