4-19-2023

eComVes: Enhancing ComVes using Data Piggybacking for Resource Discovery at the Network Edge

Sanzida Hoque
University of Missouri-St. Louis, shdcm@umsystem.edu

ECOMVES: ENHANCING COMVES USING DATA PIGGYBACKING FOR RESOURCE DISCOVERY AT THE NETWORK EDGE

by

Sanzida Hoque

A Thesis
Submitted to The Graduate School of the
University of Missouri-St. Louis
in partial fulfillment of the requirements for the degree
Master of Science
In
Computer Science

August, 2023

Advisory Committee:

Abde Mtibaa, Ph.D.
(Chairperson)

Sharlee Climer, Ph.D.

Badri Adhikari, Ph.D.

ABSTRACT

Over the past few years, Augmented Reality (AR) and Virtual Reality (VR) have emerged as highly popular technologies that demand rapid and efficient processing of data with low latency and high bandwidth, in order to enable seamless real-time interaction between users and the virtual environment. This presents challenges for network infrastructure design, which can be addressed through edge computing. However, edge computing also presents challenges, such as selecting the appropriate edge server for computing tasks in dynamic networks with rapidly changing resource availability. Named Data Networking (NDN) is a potential future Internet architecture that could provide a balanced distribution of edge services across servers, thereby preventing service disruptions. In this study, *eComVes*, a novel strategy that enhances ComVes, is proposed for information-centric edge applications that adopt a correction mechanism to ensure service execution on the highest resourced server. This mechanism allows users and intermediate routers to learn about the servers' resource status directly from the server without using any explicit control messages or probing. We evaluated the performance of eComVes against ComVes and observed an improvement in the success ratio while maintaining a consistent response time, indicating an improvement in load balance across the servers.

Keywords: Named-data Networking, edge computing, resource discovery, service orchestration.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my research advisor, Dr. Abde Mtibaa for his invaluable guidance and support. Completion of this study would not be possible without his expertise. Besides my advisor, I would like to thank the rest of the members of my advisory committee Dr. Sharlee Climer, and Dr. Badri Adhikari for taking the time to read my thesis, and for their encouragement and support.

TABLE OF CONTENTS

DEFINITION OF TERMS

capacity The maximum number of tasks that can be executed concurrently on a given server. 9–11

face A shortened form of interface; A connection point on a networking device or node, which allows it to communicate with other devices or nodes in the network. viii, 3, 4, 8–13

load Number of tasks being executable at a given server. vii, 8, 11, 12

NACK A shortened form of Negative Acknowledgment; a message that the server sends to the user to let it know whether it has received a data packet correctly or incorrectly. 5, 16

overhead Additional data transmitted over a network beyond the actual data that needs to be transmitted. 8

resource The hardware and software components of a server, are essential for any task execution. ii, 1, 5, 9, 11

service A function or program offered by a server to carry out a task to provide specific functionality. ii, 1, 3, 7, 8, 12

task A specific process that a server performs to provide a specific service; an instance of service. ii, 1, 5, 6, 8, 10, 11

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AR Augmented Reality

ICN Information Centric Networking

IoT Internet of Things

NDN Named Data Networking

VR Virtual Reality

CHAPTER 1: INTRODUCTION

The proliferation of mobile end-user devices and the Internet of Things (IoT) has given rise to diverse delay-sensitive and computation-intensive services, such as augmented reality-aided navigation and autonomous driving. These services cannot be executed by user devices with limited computational capabilities. Edge computing has been proposed to offer remote execution and while reducing communication time and network bandwidth [2].

Edge computing poses new challenges such as the selection of suitable edge servers to perform computing tasks in dynamic networks that experience frequent changes in resource availability. We refer to such server selection challenges as task orchestration. Task orchestration is critical to prevent overloading, which can lead to slow response times and/or downtime. Any interruptions or downtime in these services can result in poor user experiences and revenue losses.

Researchers have found that the Named Data Networking (NDN) architecture [1] has more potential than traditional IP (Internet Protocol) architecture to serve as a means of orchestrating tasks over edge servers [3]. NDN is a specific implementation of Information-Centric Networking (ICN) [4] where data is identified by its name, unlike traditional IP networks where data is retrieved based on the location of the data.

Most existing methods for selecting servers based on resource availability rely on network probing or control messages, which can accurately identify the highest-resourced servers but incur significant overhead [2, 5–10]. However, solutions that do not use any additional messaging are limited in their applicability and accuracy to estimate available resource. ComVes, a lightweight load-balancing strategy proposed

by Mansour et al. [11], can implicitly discover server resources without additional messaging or probing but has limitations in discovering server resources to select the highest-resourced server. In this study, we address the limitations of ComVes, namely (i) the inaccurate estimation of total upstream server capacity, and (ii) the inability to detect active traffic flow upstream. We propose a new strategy that selects the highest-resourced server based on resource availability without requiring explicit messages. This approach ensures an improved success ratio in obtaining successfully executed task data for the requested task while minimizing network resource consumption.

The rest of this thesis is as follows: Chapter 2 discussed some preliminaries of NDN and the related work on edge computing over NDN. Then Chapter 3 introduces the details of the *eComVes*. Chapter 4 discussed implementation and presented experiments and the experimental results. Finally, Chapter 5 concludes this thesis with limitations and future work.

CHAPTER 2: BACKGROUND AND RELATED WORK

In this chapter, we first present a brief introduction to NDN and then discuss existing research on task orchestration approaches in NDN.

2.1 Named Data Networking (NDN)

The Named-Data Networking (NDN) architecture, as described in [1] introduces a data-centric approach to overcome some of the limitations of host-centric IP-based networks. Instead of relying on IP addresses to identify and route data packets, NDN uses unique names or content identifiers to retrieve data from the network.

There are two types of packets in NDN: Interest and Data packets.

An Interest packet is sent by a user when it wants to retrieve data from the network. The Interest packet contains the name of the task it wants to execute, and the network uses this name to route the packet to the appropriate server node. A nonce is also included in the Interest packet, and the combination of the Service Name and Nonce ensures a distinctive identification for each Interest packet.

A Data packet, as the name suggests, contains the actual data that the user requested. It is sent by the server in response to an Interest packet. The Data packet contains the name of the data it is providing, as well as the actual content. It also includes a signature to ensure the authenticity of the data. To facilitate the forwarding of Interest and Data packets, NDN uses three main data structures:

- Forwarding Information Base (FIB): A forwarding table that maps the name prefixes of service to corresponding next-hop/outgoing faces and cost of each face. The appropriate outgoing face to forward an interest toward the producer/server(s) is determined based on this cost and the forwarding strategy

running in the node.

- Pending Interest Table (PIT): A table that keeps track of the outstanding Interests for which a router has not yet received data. The PIT is used to determine where to forward incoming data packets. When an Interest is received, a PIT entry is created that contains the Interest name, the incoming face, and a list of outgoing faces to which the Interest has been forwarded.

- Content Store (CS): A cache that stores recently received data packets. When data packets arrive at a router, they are cached in the CS. The CS is used to satisfy Interests without having to forward the request upstream to the original server. If a requested data packet is available in the CS, it can be immediately forwarded to the requesting node.

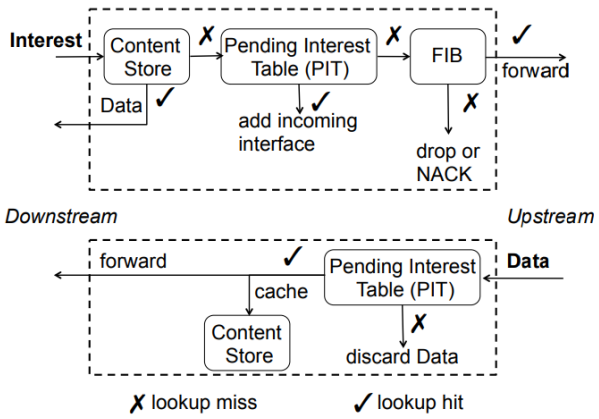


Figure 2.1: Forwarding process at a NDN node [1].

In Figure 2.1, the forwarder looks up the CS and then the PIT after receiving an interest or request. If matching Data from the CS is found, the data is returned; if a matching PIT entry is found, the incoming interface of the Interest is added to the PIT. When neither the CS nor the PIT includes a match, the forwarder records the incoming and outgoing interfaces of the Interest along with a timestamp in the PIT. The forwarding strategy then selects the output interface(s) based on the cost mapped

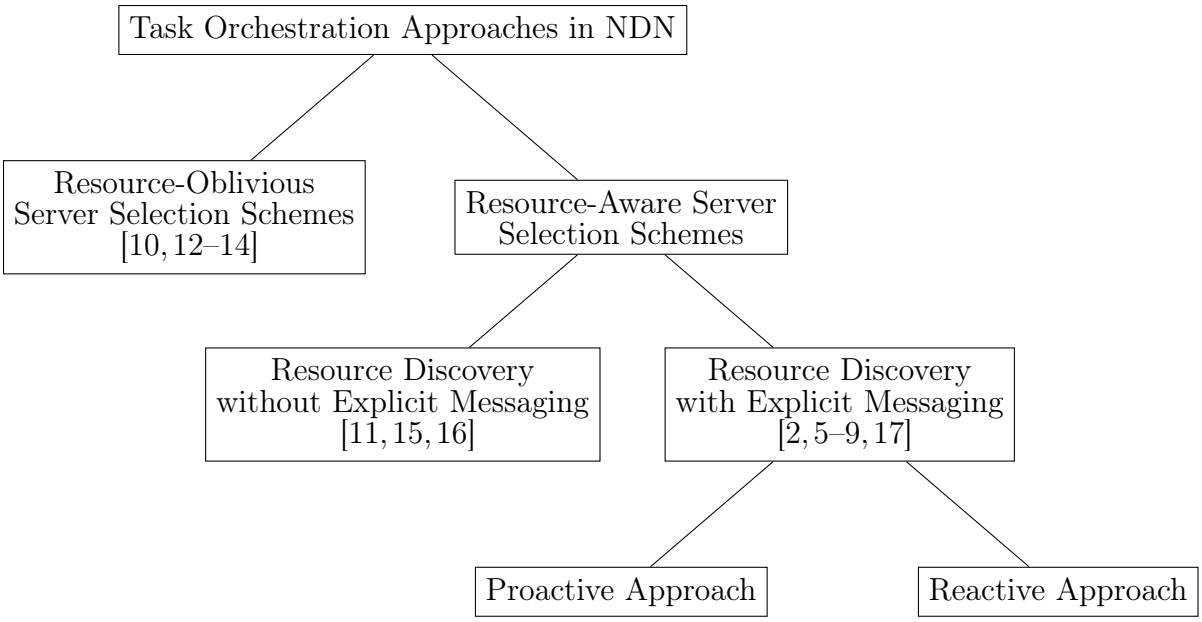


Figure 2.2: Categorization of Task Orchestration Approaches in NDN.

in the FIB. If the data is not mapped in the FIB or expired, a NACK, referred to as Negative Acknowledgment, is returned to the user.

2.2 Task Orchestration Approaches in NDN

In recent years, there has been significant research on NDN-based edge computing systems, primarily focusing on selecting servers to perform computing tasks. However, the selection process presents a significant challenge due to the rapidly changing compute resources available at edge servers. Recent works have addressed this challenge and can be categorized into two key categories (illustrated in Figure 2.2): (i) resource-oblivious server selection and (ii) resource-aware server selection.

(i) Resource-Oblivious Server Selection Schemes: Resource-Oblivious server selection schemes do not consider the resource status in the servers to select a server to execute a task. Existing solutions include Named-Function Networking (NFN) [12] and NFaaS [13]. NFN leverages function names to locate remote compute resource and perform in-network computation over NDN. Building on NFN, NFaaS [12, 13] places functions in the network and executes them through virtual machines.NDNe

[10] is another resource-oblivious server selection scheme that broadcast an Interest in searching for the optimal server based on its response time, without taking into account the resource availability of the servers which makes it a resource-oblivious scheme.

Resource-oblivious schemes have a significant limitation in that they fail to consider the servers' resource status and are prone to choosing servers with insufficient resources, which introduces additional delays.

(ii) Resource-Aware Server Selection Schemes: Resource-aware server selection schemes aim to estimate the resource status at each server to select the most suitable server to execute the task. The estimation of resource status on servers can be approached using a variety of methods, including predictive estimation based on available information or direct feedback acquisition via messages. The messaging process may be exclusively for learning resource status information, which can be referred to as explicit messaging or it might involve additional messages designed for purposes beyond resource status determination, yet effectively contributing to acquiring knowledge about resource status. Following that these schemes can further be divided into resource discovery with explicit messaging approaches and resource discovery without explicit messaging approaches.

Resource discovery with explicit messaging approaches uses additional messaging, signaling, or probing explicitly to learn about server resource status. This category can be divided into two subcategories: Proactive and Reactive approaches. In the Proactive approach, servers advertise their resource status proactively usually in a periodic manner, whereas in the Reactive approach, servers only share their status when an event occurs such as receiving a request to share the server's resource status. In their pioneering papers [18] and [19], Mtibaa et al. and Mastorakis et al. respectively highlight key challenges in NDN-based edge computing which mainly revolve around resource discovery and service discovery. Among the resource discovery with explicit

messaging approaches, Compute First Networking (CFN) [17], selects a suitable server based on the current utilization of available resources advertised by the servers, while DICer [2] proposes adapting the SVS synchronization [20] protocol to enhance NFN nodes by sharing supplementary metrics, including resource utilization to improve the service placement at task executor. Another scheme proposed by Pirmagomedov et al. [9] utilizes broadcasting an Interest through a delegated node located in the wired portion of the network on behalf of a mobile user and the delegated node gathers all servers' status from the response and selects the best server based on the collected server status. ICedge [5] is another resource discovery scheme with explicit messaging that uses additional messages to share server resource availability status, enabling the network to select the edge server based on resource availability. Kondo et al. [8] proposed a similar strategy, where servers also share resource availability status. CLedge [6], on the other hand, is a cloud-edge framework that takes into account the diverse delay requirements of services while selecting a server. Meanwhile, uDiscover [7] uses additional messages to share resource availability information, allowing the user to select the best server regarding resource availability.

Resource discovery without explicit messaging approaches for assessing resource status does not rely on receiving explicit messages or signals to learn about the state of the network. They instead use predictive analysis or leverage existing components to learn about the resource status. NDN-IoT [15] and R2 [16] are examples of resource discovery without explicit messaging approaches, but they are limited to the context of networks where servers must be in the forwarding path from the user. This strategy has a notable shortfall because it only considers the servers in the forwarding path from the user to the source of the input data for task execution, potentially ignoring other servers with more resources.

Mansour et al. proposed a resource discovery without an explicit messaging approach, named ComVes [11], which searches for the least loaded server for service

execution without using any additional control messages. This sets it apart from NDN-IoT and R2 since it makes use of all servers that are available rather than limiting itself to a certain number of servers. Each router in ComVes maintains a table named *busyness table* containing the number of pending requests each of its faces with respect to the service name which is the cost for this approach. This cost is incremented upon receipt of a new request and decremented upon receipt of a response, and it serves as the basis for searching for the best server as the number of pending interests implies the load of upstream servers. A lower cost in the table indicates a better cost. However, this strategy can not reflect on the server resource availability since it receives no explicit feedback from the server.

Explicit messaging approaches offer greater accuracy in identifying the least loaded server by obtaining feedback directly from the server, but the use of additional explicit messaging can lead to significant overhead costs. ComVes shows great potential as it can predict resource status indirectly without the need for extra messages and can function in any network. However, ComVes has its own limitations such as inaccurate estimation of upstream server capacity as well as the inability to detect active traffic flows upstream. This study aims to address these limitations and develop a strategy to overcome them.

Our study proposes a resource discovery without an explicit messaging approach that does not rely on explicit messages to learn about resource availability and instead uses piggybacking to learn about resource status in order to choose the least-loaded edge server to execute tasks. Routers get direct feedback from the servers through piggybacking and an overview of the upstream resource status and that helps to find the most resourced server.

CHAPTER 3: RESOURCE DISCOVERY AT THE NETWORK EDGE USING DATA PIGGYBACKING

In this chapter, we highlight the limitations of ComVes [11] and propose eComVes, an enhancement of ComVes to address these limitations.

3.1 ComVes Limitations

ComVes is unable to accurately estimate the resource availability of servers upstream. The forwarding based on the number of pending requests for each of its faces can lead to an inaccurate selection of the servers with the most resources available because it does not reflect the current resource availability status of the server. We have addressed **two** limitations of ComVes namely (i) the inaccurate estimation of total upstream server capacity, and (ii) the inability to detect active traffic flow upstream. Then we illustrated them in the subsequent scenarios where the problems become increasingly apparent:

3.1.1 Problem 1: Inaccurate Estimation of Total Upstream Server Capacity

ComVes estimates the resource availability using a heuristic which measures the number of pending interests in each outgoing face as an estimation of how loaded is the interface. Because the number of pending interests can indicate how much resource is used upstream but does not indicate resource availability when upstream servers have different capacities, the estimation can be inaccurate when the capacity of servers upstream are diverse. ComVes does not consider the capacity of upstream servers and assumes that all servers have the same capacity since it does not receive any direct feedback from the server. Consequently, if a given face f leads to higher available resources, but the number of the pending requests for f is higher than that of

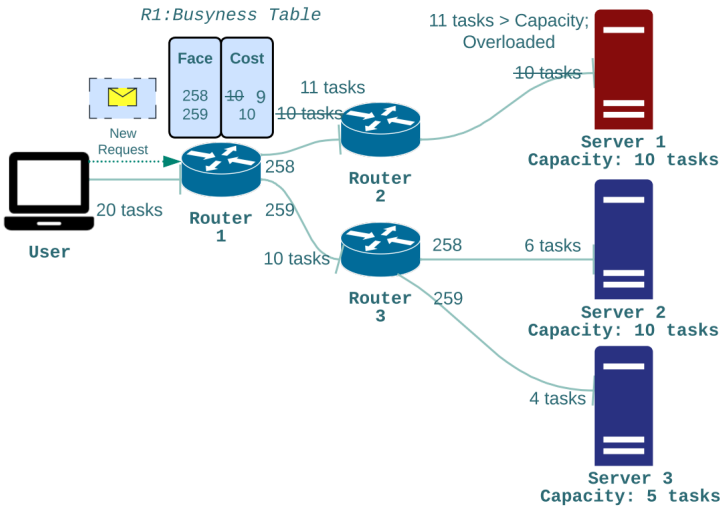


Figure 3.1: Illustration of Problem 1: Router 1, running ComVes, selects face 258 leads to Server 1 which has reached its capacity causing packet loss, despite face 259 leads to higher-resourced Server 2 and Server 3.

other faces, ComVes will still choose other faces over f which leads to higher resources and have the potential to satisfy more requests.

This problem is illustrated in Figure 3.1. The capacity of the upstream servers cannot be detected by Router 1. In the given scenario, face 258 leads to Server 1 which can handle 10 tasks, and face 259 leads to Server 2 and 3 which can handle 10 and 5 tasks respectively. Therefore, face 259 leads to more resources than face 258. But ComVes assumes that both face lead to the same amount of resources according to the cost in the busyness table and forwards request almost evenly toward those. In this scenario, Router 1 selects face 258 which will execute at Server 1. Server 1 is already overloaded; thus Interest is dropped. On the other hand, both Server 2 and Server 3 have more resources to handle the task. So, here Router 1 running ComVes is making an inaccurate forwarding decision.

3.1.2 Problem 2: Inability to Detect Active Traffic Flow Upstream

ComVes assumes that the cost of each face can be measured by only the traffic passing through the router. If there is more active traffic flow upstream using the

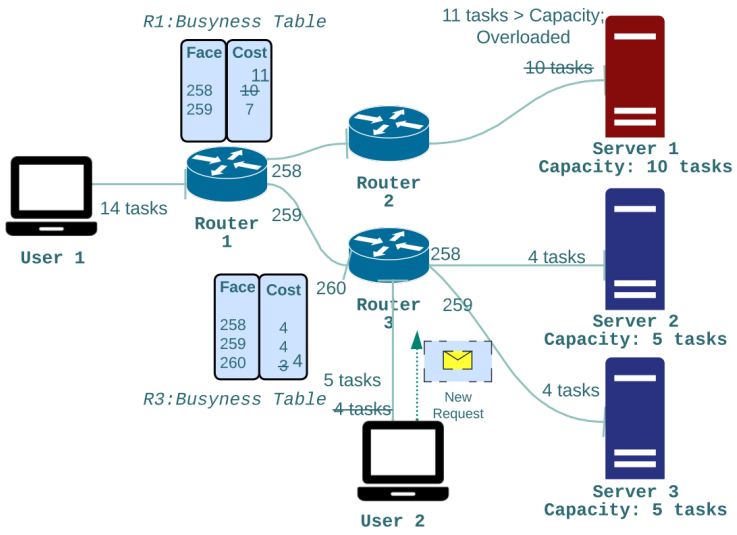


Figure 3.2: Illustration of Problem 2: ComVes at Router 3 selects face 260 leading to Server 1, despite face 258 leading to higher-resourced Server 2.

resources, ComVes fails to estimate resource consumption and fails to reflect on it while making a forwarding decision. As a result, ComVes may select a given face f with the lowest cost in the busyness table, leading to a server that is shared by additional users and, therefore, more heavily loaded, resulting in its inability to fulfill more requests than other faces. This problem is depicted in Figure 3.2. ComVes assumes that the cost associated with each face can be measured solely based on the amount of traffic passing through the router, without considering the active traffic flow of any upstream user. In the figure, receiving a new request from User 2, Router 3 with ComVes running chooses face 260 to forward the request, because it has the lowest cost of 3 in the busyness table without knowing another active traffic flow upstream from User 1 to Server 1. At router 1, even though face 259 has the lowest cost of 7, the ComVes still selects face 258 with a cost of 10, as the request cannot be forwarded downstream, resulting in a difference in cost of 4 between the two faces. The request will be directed to Server 1, which has already reached its maximum capacity, while both Server 2 and Server 3 can handle two additional tasks. As a result, ComVes makes inaccurate decisions in search of servers at the highest resource

Table 3.1: In eComVes, each router maintains a table called a *loadTable* for services. This table contains six columns namely service name, face, server hint, cost, capacity, and freshness.

Service Name	Face	Server Hint	Cost	Capacity	Freshness
/service1	258	S1	4	10	0.003200
/service1	258	S2	3	10	0.002956
/service1	260	S3	4	5	0.002897
/service2	258	S1	3	10	0.003001
/service2	260	S4	2	5	0.003256

level and cause packet loss by forwarding the request to an overloaded server.

The eComVes alleviates the problems by adding a correction mechanism that corrects the cost of each incoming face with respect to the service name on the way back of a response data reflecting on the direct feedback of servers on resource availability.

3.2 Proposed Forwarding Strategy

The proposed forwarding strategy, *eComVes*, involves a correction mechanism that corrects the cost of faces to reflect the resource availability according to the upstream servers' feedback without any explicit messaging to solve problems 1 and 2.

3.2.1 Structure and Function of *loadTable*

In this study, the proposed strategy, eComVes involves each router maintaining a table called a *loadTable* with six columns, including service name, face, server hint, cost, capacity, and freshness as shown in Table 3.1. An illustrative table is presented in Table 3.1.

The first column "Name" contains the unique name prefix of the services provided by the upstream servers, and the second column "Face" is the outgoing face toward the server of the corresponding name prefix. Each face requires to be equipped with a view of the resource status of heterogeneous servers to get a holistic perspective on the available resources upstream and thereby identify the highest-resourced server. To accomplish it, this approach employs "Server Hint" in the third column. The "Cost" in the fourth column is the main basis for forwarding decisions, which quantify the

expected number of tasks that upstream servers of each face can handle for the associated service. Therefore, a higher cost indicates a better cost. Together, these columns, "Server Hint" and "Cost", are used to monitor the resource availability of each upstream server for the associated face and the name. As having exact costs on multiple faces in the table can lead to biases and starvation of a face, the subsequent columns "Capacity" and "Freshness" are introduced. The "Capacity" column indicates the maximum number of tasks each upstream server can accommodate concurrently. The time at which the most recent task request was forwarded to a Server Hint of a Face reflects the entry's "Freshness" which is listed in the final column.

For instance, the first-row entry at the Table 3.1 signifies that Face 258 has a server upstream with the id S1 that offers service with the name prefix /service1, which can execute up to 10 tasks concurrently and is currently anticipated to execute 4 tasks with available resources, with the most recent task being forwarded from this entry at 0.003200 seconds chronologically.

3.2.2 Algorithm of the Proposed Strategy

The proposed strategy's operational procedure is presented in Algorithm 1. Initially, the loadTable of a given router R, running the proposed forwarding strategy, is populated based on the service name and the face entries of the FIB on Line 1. The Service Hint for each entry is initialized as null and the cost is initialized with a value of the estimated maximum capacity of the servers as the cost reflects the anticipated number of tasks the server can handle with available resources. However, this initialization value of cost is quickly corrected by a correction mechanism in the proposed strategy.

Upon receiving a new service request, R searches for the highest-cost entry in the loadTable for the requested service, on Line 3, to find the face that is expected to lead to the highest-resourced server. If loadTable has multiple entries of server hint per face with the highest cost, denoted as maxCost, resulting in Line 4 as true, it requires

Algorithm 1 Proposed Forwarding Strategy eComVes

INITIALIZE($loadTable, FIB$)

procedure ONRECEIVEINTEREST($interest$)
  $E \leftarrow \{e|e \in loadTable, e.cost =$ MAXCOST$(loadTable, interest.name)\ \}$
  if GETLENGTH$(E) > 1$ **then**
    $maxCost \leftarrow e.cost$
    $E \leftarrow \{e|e \in loadTable,\ e.capacity = MinCapacity(loadTable, interest.name, maxCost)\}$
    if GETLENGTH$(E) > 1$ **then**
      $minCapacity \leftarrow e.capacity$
      $E \leftarrow \{e|e \in loadTable,\ e.freshness =$ MINFRESHNESS$(loadTable, interest.name, maxCost, minCapacity)\ \}$
    end if
  end if
  FORWARDINTEREST($interest.name$, $e.face$)
  $timestamp \leftarrow$GETTIMESTAMP
  UPDATE$(loadTable, interest.name, e.face, e.serviceHint, -1, timestamp)$
end procedure

procedure ONRECEIVEDATA($data, incomingFace$)
  $serverHint \leftarrow data.metaInfo.serverHint$
  $cost \leftarrow data.metaInfo.cost$
  $E \leftarrow \{e|e \in loadTable,\ e.name = data.name\ ,\ e.face = incomingFace, e.cost = cost,\ e.serverHint = serverHint\ \}$
  if GETLENGTH$(E) >= 1$ **then**
    UPDATE$(loadTable, data.name, incomingFace, serverHint, cost)$
    while GETLENGTH$(loadTable, data.name, incomingFace) > threshhold$ **do**
      $minfreshness \leftarrow$ MINFRESHNESS(E)
      DELETE$(loadTable, data.name, incomingFace, minfreshness)$
    end while
  else
    INSERT$(loadTable, data.name, incomingFace, serverHint, cost)$
  end if
end procedure

procedure ONRECEIVENACK($nack, incomingFace$)
  $E \leftarrow \{e|e \in loadTable, e.cost =$ MINCOST$(loadTable, nack.name, incomingFace)\}$
  if GETLENGTH$(E) > 1$ **then**
    $maxFreshness \leftarrow$ MAXFRESHNESS(E)
    UPDATE$(loadTable, nack.name, incomingFace, e.hint, penaltyCost, maxFreshness)$
  else
    UPDATE$(loadTable, nack.name, incomingFace, e.hint, penaltyCost)$
  end if
end procedure

deciding which entry should be selected among all entries in set E to forward the request. We measure that the lower capacity server serves faster if two entry for two faces has the exact cost but leads to different capacity. Therefore, R looks for the face with the least capacity at Line 6 which returns all entries with maxCost and lowest capacity, denoted by minCapacity. If multiple entries with maxCost and minCapacity are returned as well, it determines which entry among those with the highest cost and lowest capacity is the oldest, having the lowest freshness at Line 9 to avoid starvation of a face. Line 9 usually returns a single entry as the time elapsed is hardly the same. When the appropriate face is identified in the loadTable associated with the requested service, R forwards the request to that face and server hint as stated in Line 12. The selected entry in the loadTable is then decremented at Line 14, as the expected number of tasks that the upstream server associated with the server hint could handle would decrease by accepting the new request. The freshness column is also updated at Line 14 with new times returned from Line 13 since Freshness keeps track of the time the most recent task request was sent, ensuring that the same entry is never chosen repeatedly and preventing bias towards one entry.

Correction Mechanism: The correction mechanism of the proposed strategy, eComVes is shown in Lines 16-29. When *R1* running the strategy receives a response Data packet on Line 16, it learns the server hint and cost information from the packet's meta info. The strategy checks if the server hint associated with the name and incoming face is in the loadTable at Line 20. If it is, it corrects the cost in the loadTable for the server hint that is associated with the face and name on Line 21. Otherwise, it inserts the information into the loadTable since the router does not have any previous entry for this service hint on Line 27. In Line 20, it also checks if the number of entries associated with the incoming face and name is greater than threshhold. If so, R deletes the oldest entry associated with the name and faces the loadTable to maintain scalability at Lines 22-25.

NACK feedback: The NACK feedback operation in the strategy is demonstrated in Lines 30-38. A Server responds with a NACK when it does not have resources available to execute the task. When R receives a NACK, the entry in the loadTable with the lowest cost service hint for the requested service name and the incoming interface downstream receives a penalty cost that is significantly lower, at Line 34, preventing R from selecting the face-leading overloaded server until it receives a data correction. An entry with the lowest cost service hint in the loadTable gets the penalty cost because an entry with the least resource is more likely to be overloaded. If multiple entries have the lowest cost, R selects the face with the newest entry as it is more prone to overload.

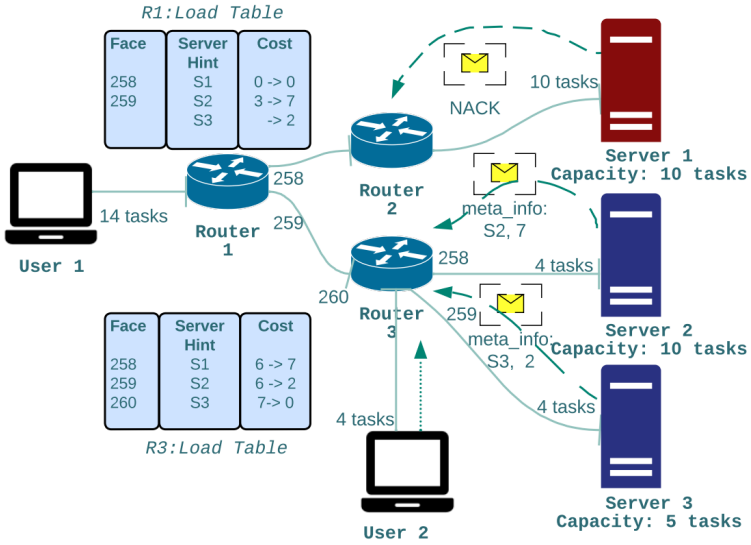


Figure 3.3: Demonstration of eComVes: meta info in data from Servers 2 and 3, and NACK from Server 1 update the cost in the loadTables of downstream routers, including Router 1 and Router 3. The reduced cost resulting from receiving NACK prevents Server 1 from becoming overloaded, while the cost and server hint from Servers 2 and 3 provide an overview of resource availability to Router 1. Therefore, when Router 3 receives a new request from User 1 via Face 258, it selects Face 258 as per the loadTable and reaches the most resourced Server 2.

3.2.3 eComVes in Action

The high-level example of the proposed strategy, eComVes is shown in Figure 3.3 illustrating how the eComVes handles problems 1 and 2. In the given scenario depicted

in Figure 3.3, Server 1 has reached its maximum capacity while Servers 2 and 3 still have the capacity to execute more tasks. When downstream routers, such as Router 1 and Router 3, receive data with meta-information from Servers 2 and 3, they update their cost calculations, which previously did not consider server capacity, and update the loadTables accordingly. As a result, the cost of Server 1, which was previously the highest at 7 for Router 3, is now set to 0, while the cost for Server 2 and Server 3 is 7 and 2, respectively, for Router 3. Previously, these costs were 6 for Router 3, and Router 1 was unaware of the existence of two servers. With this information, Router 3 can now send more requests via Face 259 and avoid Face 260, which is overloaded. This approach prevents Server 1 from becoming overloaded, and the meta-information from Servers 2 and 3 provides an overview of resource availability to Router 1.

This approach addresses the two problems of ComVes described in Section 3.1. It solves ComVes's inability to accurately estimate upstream server capacity by providing a complete overview of resource availability to downstream routers. Therefore, Router 1 can now select the highest-resourced server, which is Server 2 in this case. It solves ComVes's inability to detect upstream active traffic flow as well by sending a reduced cost with the receipt of NACK. Therefore, Router 3 does not choose Server 1, because of its highest resource score estimated using explicit overload NACK message.

CHAPTER 4: RESULTS

The aim of this section is to assess the effectiveness of the proposed strategy, eComVes using a simulation study. Specifically, we seek to determine: (i) if eComVes selects the most highly-resourced server through task distribution, (ii) if eComVes achieves better success of satisfying requests overall using success ratio metric (iii) and, if eComVes leads to a decrease in overall response time through CDF of delays.

4.1 Implementation

Every router runs the eComVes and maintains a loadTable. The loadTable updates upon receiving a request, data, or NACK according to the algorithm 1. A point to be noted is that algorithm 1 Line 24 is included in the design, but it is not part of this implementation. The server application sends data with Metainfo carrying server hint which is the unique ID of the corresponding server and the cost, which is the current number of tasks it can accommodate, calculated from the difference of capacity and current load of a server. It estimates the computation time using the function described in the next paragraph and schedules the task to be completed within that time and sends NACK if the estimation says it can accommodate the new request. We implement ComVes [11] and compare its performance against the eComVes's performance.

We use the following function for estimate the task execution time at server, $T(i) = \frac{u_i}{c_i} + \alpha$, where u_i represents the resource utilization of server i, c_i denotes the capacity of the server i, and α stands for an insignificantly small randomization factor for requested service, accounting for any additional time requirements. This randomization factor is added to the function as we aim to replicate the real-world scenario on a

Table 4.1: Simulation parameters

Parameters	**Value(s)**
number of servers	{2, 3}
number of users	{1, 2}
number of services	1
frequency (request per second)	{25, 30, 35, 40, 45}
capacity (number of tasks)	[3 - 60]
link delay (ms)	10
link bandwidth (Mbps)	1
simulation time (seconds)	50

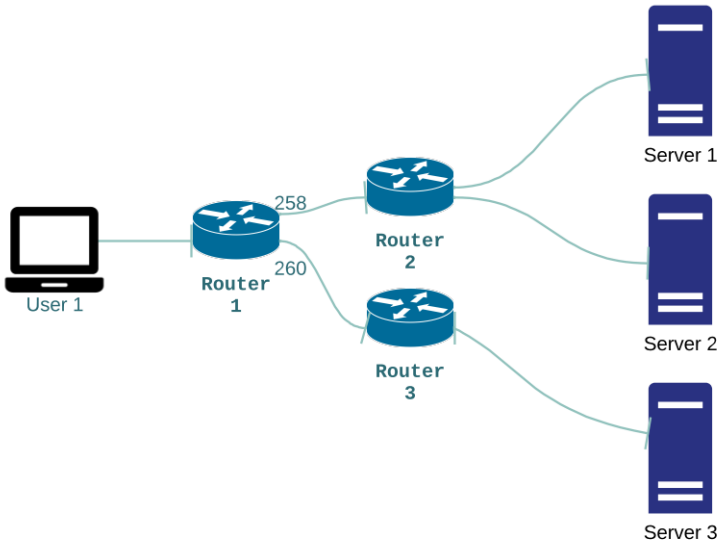
smaller scale where there can be many factors that can affect the response time. This function serves as a fundamental model for estimating task request response time. This function returns higher delays for higher-loaded servers.

4.2 Experimental Setup

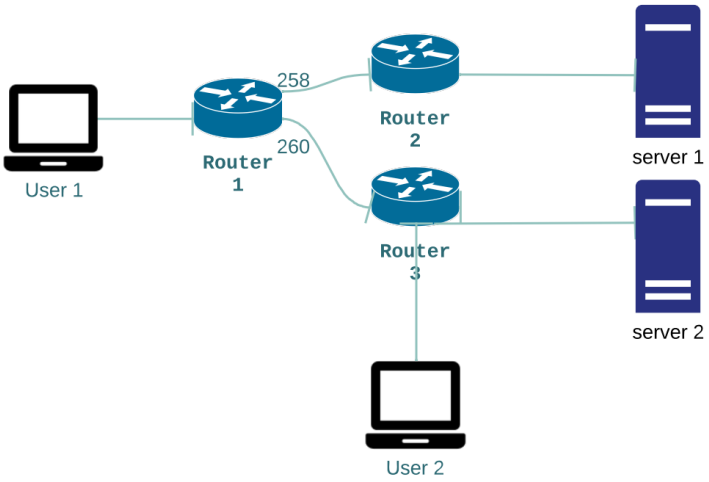
We implement and evaluate the strategy described in Section 3.2. Using ndnSIM [21] module within ns-3, the simulation was conducted on a Desktop machine equipped with a 7-core Intel CPU and 64 GB of memory.

We use two simple network topologies shown in Figure 4.1 for our simulations. Figure 4.1(a) is the topology used to evaluate the performance of the eComVes for problem 1, and Figure 4.1(b) is the topology used for problem 2.

Table 4.1 summarizes the full simulation parameters used in our evaluation. To observe the performance of the two strategies, we conduct simulations that measures response time and the task distribution under varying frequency (request rates). The bandwidth and link delay are set to 1 Mbps and 10ms. The generated results are averages of 5 runs and each simulation lasts for 50 seconds. The experiment is configured for a single service. Therefore, the users send only tasks belonging to one service. We will extend the work to support multiple services in the future.In both topologies, User 1 sends requests at a rate of 25, 30, 35, 40, or 45 tasks per second. We refer to such request rates as frequency shown in Table 4.1. ComVes and the



(a) Evaluation Topology for Problem 1: User 1 sends request at rate of 25, 30, 35, 40, 45 per seconds; capacities of Server 1 and Server 2 is 20 tasks and Server 3 is 5 tasks.



(b) Evaluation Topology for Problem 2: User 1 sends request at rate of 25, 30, 35, 40, 45 per seconds and request rate of User 2 is three times that of User 1; capacities of Server 1 and Server 2 is equal in the range of 36 tasks each.

Figure 4.1: Evaluation topology for Problem 1 and Problem 2.

eComVes are limited to discovering upstream servers up to 2 hops. This limitation is reflective of actual operational conditions in real-world scenarios, and our simulation aims to replicate this scenario on a smaller scale.

The first topology in Figure 4.1(a) for problem 1 includes three servers, where both Server 1 and Server 2 have a capacity of 20 tasks and Server 3 has a capacity of 5 tasks. Router 1 is linked to one user having two upstream interfaces (258, 260) where face 258 leads to Server 1 and 2 and 260 leads to Server 3.

The second topology for problem 2 is presented in Figure 4.1(b) where there are two users and two servers included.In this topology, User 1 is unaware of the active link of User 2 upstream due to limiting hop. The frequency of User 2 is three times that of User 1. All the servers (Server 1 and Server 2) have an equal resource capacity of 36 tasks to observe the impact of problem 2.

4.3 Evaluation Metrics

We consider the following metrics to evaluate our proposed solution:

1. Task Distribution: The number of tasks received by each server. We differentiate the number of processed tasks scheme (processed) and the number of failed tasks scheme (overloaded) due to the overloaded server.

2. Success Ratio: The ratio between the total number of satisfied requests received by the user and the total number of requests sent by that user.

3. Request Response Time: Time elapsed between a user sending a request and receiving a response from the server.

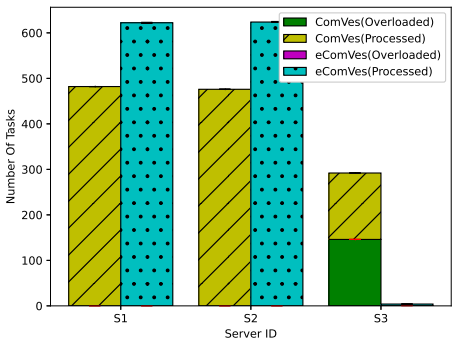
4.4 Experimental Result

In this section, we analyze the results and compare the performance of the eComVes with ComVes.

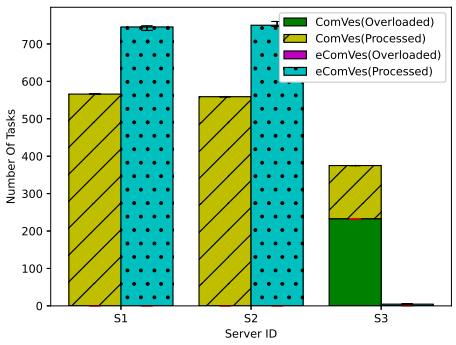
4.4.1 Result and Analysis for Problem 1

This subsection discusses the results for Problem 1 described in the topology presented in Figure 4.1(a). We compare the performance of the eComVes, and those of ComVes.

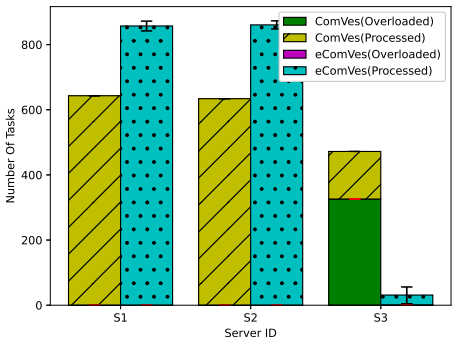
Task Distribution: First, we measure the task distribution across servers (S1, S2, S3) to visualize the overall selection mechanism of eComVes compared with ComVes. This task distribution is presented in Figure 4.2 using a bar chart where pattern bars indicate the executed task and the solid part indicates the task failed due to an overloaded server. The green and yellow bars present results for ComVes, whereas the purple and cyan represent eComVes. Figure 4.2(a) with frequency 25 shows that ComVes has about 12% overload occurrence with almost one-fourth of the total requests sent to Server 3 with lower resources, while eComVes has an occurrence of the overloaded server of almost 0% with just a few tasks sent to Server 3. Server 3 receives very few tasks because Router 1 running eComVes receives the corrected cost with server hint piggybacked on the response data, and due to the difference in cost, the cost for servers 1 and 2 never drops below Server 3 in loadTable. Requests are promptly handled by servers 1 and 2, preventing the cost from falling below the highest cost observed with Server 3. With a lower frequency in comparison to the total capacity of upstream servers, servers can accommodate all the requests. As we increase the frequency of requests, the task distribution gets critical, and the wrong decision regarding selecting the highest-resourced server results in more packet loss due to an overloaded server. Therefore, in Figure 4.2(b) with frequency 30, the rate at which servers experience overloads for ComVes is increased to 16% higher than that observed with eComVes. In Figure 4.2(c) with frequency 35, around 27% of requests are sent to Server 3 for ComVes while Server 3 starts receiving around 1% of requests for eComVes at this point, which results in 19% more successful execution of requests on servers than ComVes. This trend continues at Frequency 40 as shown



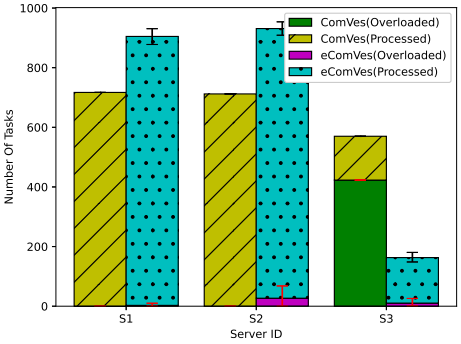
(a) Task Distribution for frequency 25.



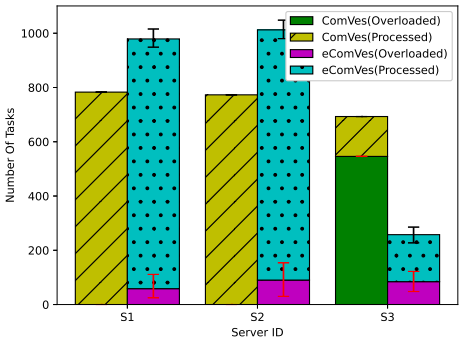
(b) Task Distribution for frequency 30.



(c) Task Distribution for frequency 35.

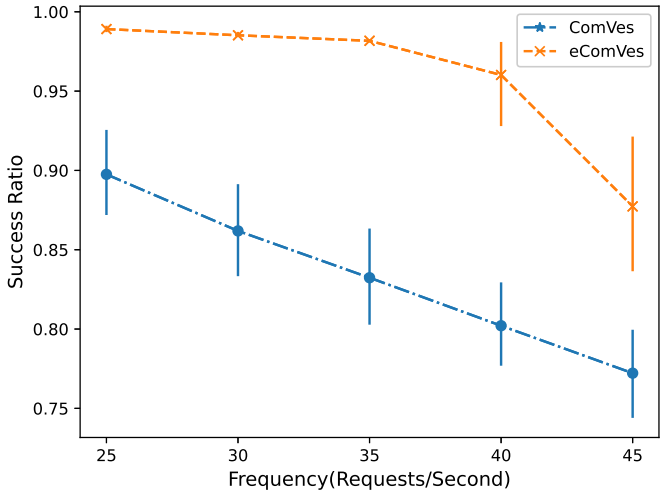


(d) Task Distribution for frequency 40.

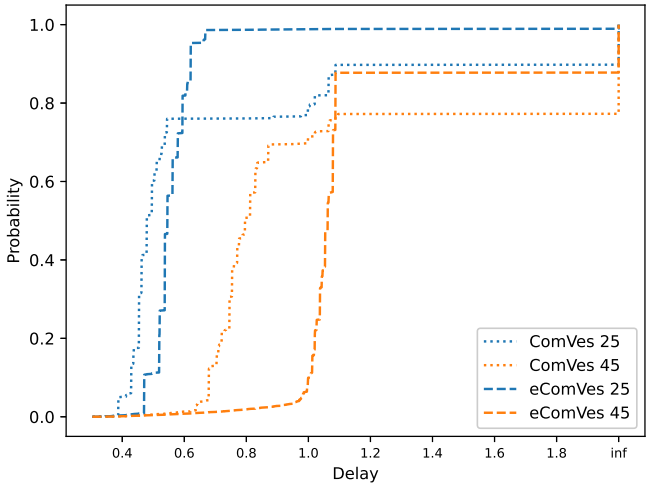


(e) Task Distribution for frequency 45.

Figure 4.2: Comparing impacts of service request rate (frequency) on task distribution for Problem 1.



(a) Success Ratio



(b) CDF of Response Times

Figure 4.3: Comparing the impact of service request rate on success ratio and response time for Problem 1.

in Figure 4.2(d) where ComVes has about 21% overload occurrence with almost 29% of the total requests sent to Server 3, while the eComVes has the occurrence of the overloaded server of less than 1% with 8% tasks sent to Server 3. This difference in the experiencing overloaded servers decreases slightly, which is around 20% at Frequency 45 in Figure 4.2(e) as the cost becomes 0 or penalty cost for all faces in the loadTable at some point when the frequency is higher exceeding the total capacity of the upstream server. As the number of tasks increases in the network, all servers become overloaded.

Success Ratio: Figure 4.3(a) shows the ratio of user 1 requests successfully satisfied by any server at different rates to the total requests sent. This figure shows that the success ratio of both eComVes and ComVes decreases as frequency increases. At a lower request rate, most of the requests can be accommodated, and the success ratio for both strategies can get close to 1, resulting in an insignificant gain for eComVes. Therefore, the cost of the faces in loadTable is always greater than zero for eComVes. In our simulation, frequency 25 is the lowest frequency, and eComVes achieved a success ratio of almost 1, 10% more than ComVes. As the frequency increases, the lowest-loaded server selection becomes critical. Therefore, the success gain of eComVes. which can be defined as the success ratio difference between eComVes with Comves, increasing to almost 13% at frequency 30 and continuing to increase to 19% at frequency 35 with a success ratio of 1. When the frequency reaches 40, servers start to experience overloaded, resulting in cost 0 or penalty cost in the loadTable and the gain decreases to 10% for eComVes. This is because, at a higher frequency, both strategies fail to accommodate requests due to resource unavailability. The success gain decreases more at the frequency 45 to 5% and becomes the lowest gain in the simulation. According to the simulation results, frequency 35 is the sweet spot for our experiment, where eComVes performs best, outperforming ComVes by 19% in the success ratio. After this point, the gain starts to decrease with increasing

frequency.

Request Response Time: To investigate the impact of different task-sending rates on response time, we compare the CDF of eComVes response time and ComVes response time. We present the CDF for frequencies 25 and 45. In the figure, we refer to "scheme XX" where the scheme is ComVes or eComVes and XX is the sending rate (25, 30, 35, 40, or 45).

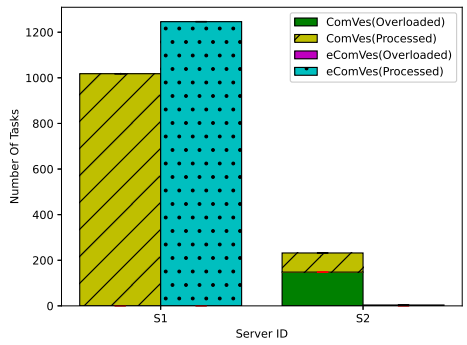
Figure 4.3(b) shows that the response time of eComVes is mostly less than 0.7 seconds for frequency 25. There are less than 1% overload occurrences, denoted by Inf (infinity). More than 99% tasks had response times within 0.7 seconds, resulting in one cluster. eComVes balances the load across the servers, and one cluster indicates a balanced load across servers and consequently stable delays, whereas ComVes has a gradually increasing curve, indicating that a wide range of response times are covered. This suggests that some tasks took longer to complete than others, resulting in a spread-out CDF curve. This is a result of Server 3 receiving more requests in ComVes, which results in either longer execution times or overloaded servers. For ComVes 10% of tasks have an overload occurrence, whereas for eComVes, almost 0% of tasks experience overload occurrence.

For frequency 45, delays are larger due to a higher load of requests. The percentage of reaching infinity increases for both strategies as the higher frequency degrades both performances, resulting in a lower gain for eComVes. Similar to frequency 25, eComves's clustered value indicates more consistent performance within a specific range, while ComVes's scatter value suggests a wider variability in performance affected by load imbalance. ComVes has fluctuations with multiple clusters, whereas eComVes has consistent delays with one cluster, implying better load balancing across the server.

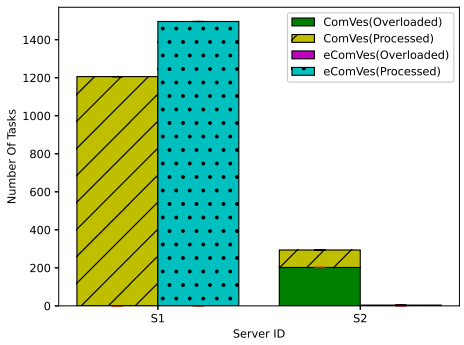
4.4.2 Result and Analysis for Problem 2

It is essential to note that, the performance evaluation results for Problem 2 are specific to the requests from User 1 only, as all requests from User 2 are directed to Server 2 exclusively due to a common hop limit imposed on both strategies discussed earlier.

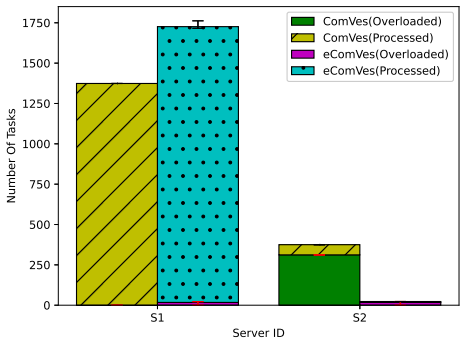
Task Distribution: In Figure 4.4, we present the task distribution, which includes processed and dropped tasks due to overloaded servers, on two servers with different traffic flows for varying request rates to compare the eComVes with ComVes for Problem 2. Figure 4.4(a) with frequency 25 shows that eComVes has almost no overload occurrence with only a few tasks sent to Server 2, whereas ComVes chooses overload server of around 12% of the time with almost one-fourth of the total requests from User 1. This is because R1 running ComVes assumes from its busyness table that the requests from User 1 are only being forwarded toward the upstream server. It almost evenly distributes all the requests toward two faces and cannot see the active traffic link in the upstream that consumes resources on Server 2. As we increase the frequency in ComVes, the server experiences 13% overload occurrence at frequency 30 forwarding around 20% of total requests from User 1 to Server 2, and 18% overload occurrence at frequency 35 forwarding around 21% of total requests from User 1 to Server 2. Given that, for eComves the server experiences almost no overload and forwards almost no requests to server 2. Increasing the frequency more for eComVes, Server 2 also starts receiving requests from User 1; 6% of requests from User 1 are forwarded to Server 2 at frequency 40, and almost 11% are forwarded to Server 2. This causes both servers to be overloaded by 12% at frequency 40 and 21% at frequency 45. This is because, at this point, the request rate exceeds the total upstream server capacity and server selection becomes random. In contrast, in ComVes, 24% and 27%, respectively, of requests are forwarded to server 2 at frequencies 40 and 45. This results in 23% overload at frequency 40 and 27% overload at



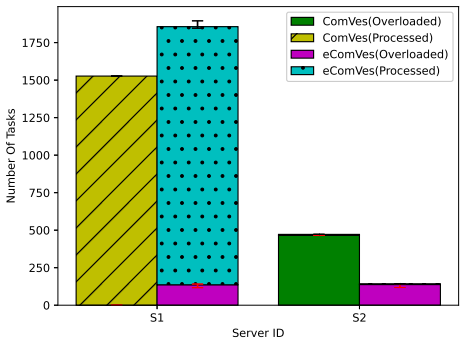
(a) Task Distribution for frequency 25.



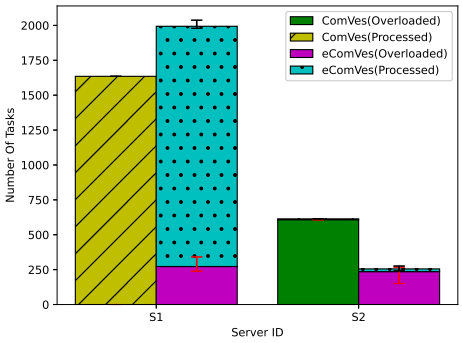
(b) Task Distribution for frequency 30.



(c) Task Distribution for frequency 35.

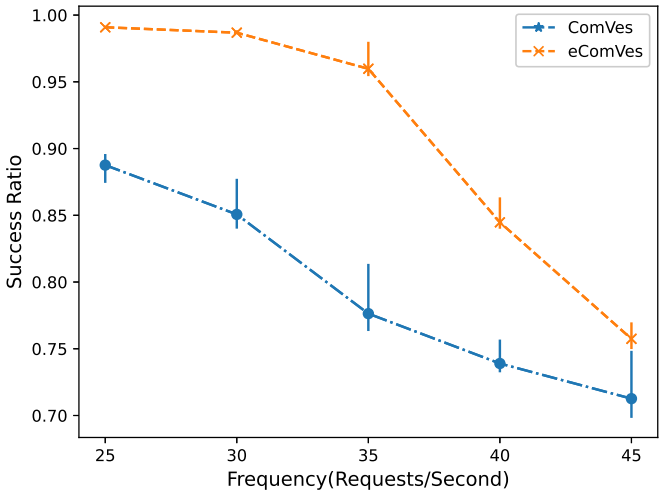


(d) Task Distribution for frequency 40.

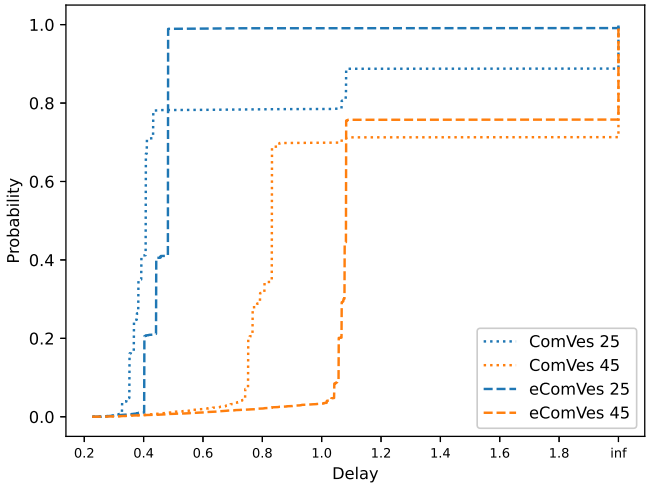


(e) Task Distribution for frequency 45.

Figure 4.4: Comparing impacts of service request rate (frequency) on task distribution for Problem 2 (Specific to request from User 1).



(a) Success Ratio



(b) CDF of Response Times

Figure 4.5: Comparing the impact of service request rate on success ratio and response time specific to requests from User 1 for Problem 2.

frequency 45. R1 running ComVes forwards more requests to Server 2 than eComVes because ComVes does not distinguish the difference in capacity in the upstream server, whereas eComVes receives real-time resource availability from the upstream server.

Success Ratio: In Figure 4.5(a) eComves perceives the difference as it receives direct feedback from servers on resource availability with server hints piggybacked on the Data. Therefore, it avoids going to Server 2, as the cost never gets higher than the face towards Server 1 due to mostly consumed resources. Due to the identical cause discussed earlier, the success ratio in Figure 4.4 also corresponds to User 1's requests only.

Figure 4.5(a) shows the ratio of user requests from User 1 successfully satisfied at different rates to the total requests sent and the impact of the user request rate on this ratio. This figure shows that the success ratio of both eComVes and ComVes decreases as frequency increases. This is because, at a very high frequency, the request rate becomes higher than the total capacity of the upstream server, and all servers become overloaded at a certain point. As there are no resources left on any server, the cost for each face in loadTable becomes 0 or penalty cost resulting in a random server selection. Therefore, the success ratios of both strategies go downward as both fail to execute requests due to resource unavailability. In our simulation, the highest frequency we set is 45 tasks per second, where the total capacity of upstream servers is 25. eComVes still manage to complete about 5% more requests successfully before all servers become overloaded. As we decrease the frequency, eComVes becomes almost 10% better than ComVes in terms of successful request execution at frequency 40 and achieves 100% success ratio at frequency 35 whereas ComVes fails requests almost 19% time due to server overload. eComVes achieves 100% success ratio at rates of 25 and 30 and fails around 13% at frequency 30 and 10% at frequency 25. Comparing ComVes itself to frequency 30, ComVes performs slightly better at frequency 25. This is because server 2 is relatively less loaded at 25 than it is at 30, and choosing a

server has less of an impact with less load. Based on the simulations, we discover that frequency 35 is the sweet spot for eComVes, where it outperforms ComVes most by 18% in the success ratio.

Request Response Time: The CDF in Figure 4.5(b) illustrates the response time for Problem 2. For frequency 25, 20% of the tasks have longer delays of more than 1 second in ComVes. This happens because ComVes forwards requests to server 2, which is loaded with requests from user 2. These requests either fail due to an overloaded server indicated at the infinity point or execute with a high delay, causing the outlier cases. On the other hand, eComVes demonstrates consistent performance in more than 99% of tasks, and less than 1% of tasks go to infinity due to an insignificant number of packet losses. At frequency 45, servers encounter resource unavailability, leading to a cost of 0 or a penalty cost for every face in the loadTable. The delay for both strategies increases with frequency. However, eComVes can execute a greater number of requests at least 5%(for frequency 45), and shows a consistent response time where ComVes has multiple clusters, including higher delays.

It is essential to recognize that this experiment has some limitations, such as the lack of multiple services, running on simple topologies, and the exclusion of mobility factors. However, these limitations serve as a starting point for further research.

CHAPTER 5: CONCLUSIONS AND FUTURE WORKS

In this study, we introduce a resource discovery scheme called *eComVes* for information-centric edge applications that utilizes a correction mechanism to select the highest-resourced edge servers for the execution of the service. The proposed mechanism allows users and intermediate routers to learn about the status of edge servers directly from the servers' feedback, eliminating the need for explicit control messages or probing.

Our simulation findings show that eComVes improves in terms of success ratio, having a maximum gain of 19% for problem 1 and 18% for problem 2 against ComVes while maintaining consistent response time, indicating an improvement in load balance across the servers. We plan to investigate the eComVes in scenarios that involve multiple users and multiple services in complex and larger topologies.

REFERENCES

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[2] U. Ambalavanan, D. Grewe, N. Nayak, L. Liu, N. Mohan, and J. Ott, "Dicer: distributed coordination for in-network computations," in *Proceedings of the 9th ACM Conference on Information-Centric Networking*, pp. 45–55, 2022.

[3] D. Sabella, A. Alleman, E. Liao, M. Filippou, Z. Ding, L. G. Baltar, S. Srikanteswara, K. Bhuyan, O. Oyman, G. Schatzberg, *et al.*, "Edge computing: from standard to actual infrastructure deployment and software development," *ETSI White paper*, pp. 1–41, 2019.

[4] X. Jiang, J. Bi, G. Nan, and Z. Li, "A survey on information-centric networking: rationales, designs and debates," *China Communications*, vol. 12, no. 7, pp. 1–12, 2015.

[5] S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, "Icedge: When edge computing meets information-centric networking," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4203–4217, 2020.

[6] M. W. Al Azad, S. Shannigrahi, N. Stergiou, F. R. Ortega, and S. Mastorakis, "Cledge: A hybrid cloud-edge computing framework over information centric networking," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pp. 589–596, IEEE, 2021.

[7] G. Torres, R. Tourani, A. Mtibaa, D. Stelmakh, S. Misra, S. Srikanteswara, Y. Zhang, and S. Hoque, "udiscover: User-driven service discovery in pervasive edge computing using ndn," in *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*, pp. 77–82, IEEE, 2022.

[8] D. Kondo, T. Ansquer, Y. Tanigawa, and H. Tode, "Resource discovery for edge computing over named data networking," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 552–559, IEEE, 2021.

[9] R. Pirmagomedov, S. Srikanteswara, D. Moltchanov, G. Arrobo, Y. Zhang, N. Himayat, and Y. Koucheryavy, "Augmented computing at the edge using named data networking," in *2020 IEEE Globecom Workshops (GC Wkshps*, pp. 1–6, IEEE, 2020.

[10] M. Amadeo, C. Campolo, and A. Molinaro, "Ndne: Enhancing named data networking to support cloudification at the edge," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2264–2267, 2016.

[11] D. Mansour, H. Osman, and C. Tschudin, "Load balancing in the presence of services in named-data networking," *Journal of Network and Systems Management*, vol. 28, pp. 298–339, 2020.

[12] C. Tschudin and M. Sifalakis, "Named functions and cached computations," in *2014 IEEE 11th consumer communications and networking conference (CCNC)*, pp. 851–857, IEEE, 2014.

[13] M. Król and I. Psaras, "Nfaas: named function as a service," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 134–144, 2017.

[14] M. Król, K. Habak, D. Oran, D. Kutscher, and I. Psaras, "Rice: Remote method invocation in icn," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pp. 1–11, 2018.

[15] M. Amadeo, G. Ruggeri, C. Campolo, and A. Molinaro, "Iot services allocation at the edge via named data networking: From optimal bounds to practical design," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 661–674, 2019.

[16] J. Qi and R. Wang, "R2: A distributed remote function execution mechanism with built-in metadata," *IEEE/ACM Transactions on Networking*, 2022.

[17] M. Król, S. Mastorakis, D. Oran, and D. Kutscher, "Compute first networking: Distributed computing meets icn," in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pp. 67–77, 2019.

[18] A. Mtibaa, R. Tourani, S. Misra, J. Burke, and L. Zhang, "Towards edge computing over named data networking," in *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 117–120, IEEE, 2018.

[19] S. Mastorakis and A. Mtibaa, "Towards service discovery and invocation in data-centric edge networks," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–6, IEEE, 2019.

[20] P. Moll, V. Patil, N. Sabharwal, and L. Zhang, "A brief introduction to state vector sync," *Named Data Networking, Tech. Rep. NDN-0073, Revision*, vol. 2, 2021.

[21] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnsim: An open-source simulator for ndn experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 19–33, 2017.