Enhancing Wireless Network Performance through Advanced Caching Techniques

Major Project Report Part 1 / Part 2

Submitted by

Aadel Abdul Sammad

(B200889EC)

Safa Saliha

(B200025EC)

Hamdha Abdul Rasheed KK

(B200875EC)

Nirupam M Suraj

(B200901EC)

In partial fulfillment for the award of the Degree of

BACHELOR OF TECHNOLOGY IN
ELECTRONICS AND COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY, CALICUT

NIT CAMPUS P.O., CALICUT

KERALA, INDIA 673601.

NATIONAL INSTITUTE OF TECHNOLOGY, CALICUT

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

*This is to certify that the major project report entitled "**Enhancing Wireless Network Performance through Advanced Caching Techniques**" is a bonafide record of the Project done by **Aadel Abdul Sammad** (B200889EC),**Hamdha Abdul Rasheed KK** (B200875EC),**Nirupam M Suraj** (B200901EC),**Safa Saliha** (B200025EC) under our supervision, in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Electronics and Communication Engineering** from **National Institute of Technology Calicut**, and this work has not been submitted elsewhere for the award of a degree.*

Dr. Nujoom Sageer Karat
Assistant Professor ECED,
NIT Calicut

Dr. Jaikumar M.G
Head Of Department ECED,
NIT Calicut

Place: Calicut

Date: 07 - 05 - 2024

ACKNOWLEGEMENT

We would like to convey our sincere regards to everyone who has supported us throughout the course of this project, for helping and motivating us in completing the same. We express our heartfelt gratitude to Dr. Nujoom Sageer Karat, our project guide, for mentoring and guiding us through this project with his valuable insights, without which, we would not have been able to complete the project successfully. This work was supported in part. We would like to thank the Project Co-ordinator Dr Suresh Rangan, and the evaluation committee for all the suggestions put forward during the project evaluations. We would like to thank Department of Electronics and Communication Engineering, National Institute of Technology Calicut, for providing us with an opportunity to work on this project and also for its unwavering support during the entirety of the project, which has significantly contributed to the project coming to fruition

ABSTRACT

In the context of wireless caching networks, the need to enhance cache hit rates based on content popularity becomes increasingly crucial in light of rising data traffic demands. The paper focuses on fundamental wireless caching network scenario, involving a source server linked to a base station (BS) serving numerous user requests. The issue can be addressed by creating an innovative strategy for dynamic content updates, exploiting deep reinforcement learning (DRL). Given the base station's limited knowledge of content popularity, this strategy dynamically refreshes the BS cache based on evolving requests and the current cache contents. Through simulations the project aims to demonstrate superior performance by yielding higher average rewards and achieving a better cache hit rate than existing replacement policies, including Least Recently Used (LRU), first-in first-out(FIFO), and deep Q-network-based algorithms. The paper then delves into an innovative approach aimed at reducing transmission rates, compared to the caching scheme proposed in [6]. Unlike the scheme in [6], which operates under the assumption of equal content popularity, our focus is on accounting for varying content popularities to devise a novel strategy.

CONTENTS

List of Figures

List of Tables

List of Abbreviations

BS	Base station
DRL	Deep reinforcement learning
LRU	Least Recently Used
FIFO	First-in first-out
APs	Access points
DQN	Deep Q-Network
Iot	Internet of Things
RL	Reinforcement learning
CDNs	Content delivery networks
DNN	Deep neural network

CHAPTER 1

Introduction

In recent years, there has been a significant rise in the usage of mobile data. This increase is likely to put a lot of pressure on wireless networks in the coming years. As a result, these networks might become crowded and struggle to provide good service quality for users. Caching is a technique to reduce peak traffic rates by prefetching popular content in memories at the end users [1]. It is used to temporarily store data, usually coming from an origin source, within a memory that is quickly accessible. Caches decrease access time as repeated requests for the same data are served by a fast/local memory rather than by a remote source. Wireless edge caching is a specific type of caching method that involves storing and delivering frequently requested content or data at the edge of a wireless network, typically at base stations (BSs) or access points (APs) that are close to the end-users. Cache hit rate, also known as cache hit ratio, is a critical performance metric used in computing and data management to measure the efficiency of a cache system. Cache hit rate is defined as the proportion or percentage of memory accesses that successfully retrieve data from the cache, as opposed to accessing the main or slower storage. A high cache hit rate indicates that a significant portion of memory accesses are being satisfied from the cache, resulting in improved system performance due to reduced access latency. Efficient cache management strategies, such as algorithms for cache replacement (e.g., LRU - Least Recently Used) and cache prefetching, are generally employed to maximize the cache hit rate and enhance system performance in various computing environments

1.1 Reinforcement learning in Caching System

The caching system employs a reinforcement learning (RL) agent to dynamically adapt its caching strategy based on real-time system observations. Unlike traditional caching algorithms that rely on predefined rules or static models, the RL agent actively explores the environment to learn the expected utility of each action in each state [2]. This value-based approach enables the agent to continuously refine its caching policies and optimize system performance. To enhance the adaptive caching system, the RL agent utilizes a feature-based variant of Q-Learning, a value-based method that avoids storing all possible Q-value pairs. Instead, Q-values are computed using real-time state features, enabling efficient learning even in large state and action spaces. Deep Q-Network (DQN) is employed for efficient learning, updating Q-values in batches rather than iteratively. This approach improves computational efficiency and enables the incorporation of more complex state representations. A memory stores past transitions for delayed training, allowing the agent to learn from past experiences. DQN also uses a technique called double Q-learning to improve its stability. Double Q-learning involves using two separate sets of Q-values, one for evaluation and one for learning. This helps to prevent the agent from overestimating the value of its actions. As it explores, it updates its understanding of the values of each action until it eventually learns the optimal policy. In addition to exploration, DQN also uses a technique called exploitation to balance exploration with taking the best action it knows of. Exploitation is simply taking the action that has the highest value according to the agent's current understanding. The reinforcement learning algorithm is able to learn from a suboptimal policy and explore with lower overall regret.

The implementation of reinforcement learning in the adaptive caching system involves careful selection of hyperparameters, which influence the agent's learning behavior and its ability to balance exploration and exploitation. Their optimal values depend on the specific characteristics of the caching system and the workload it serves.

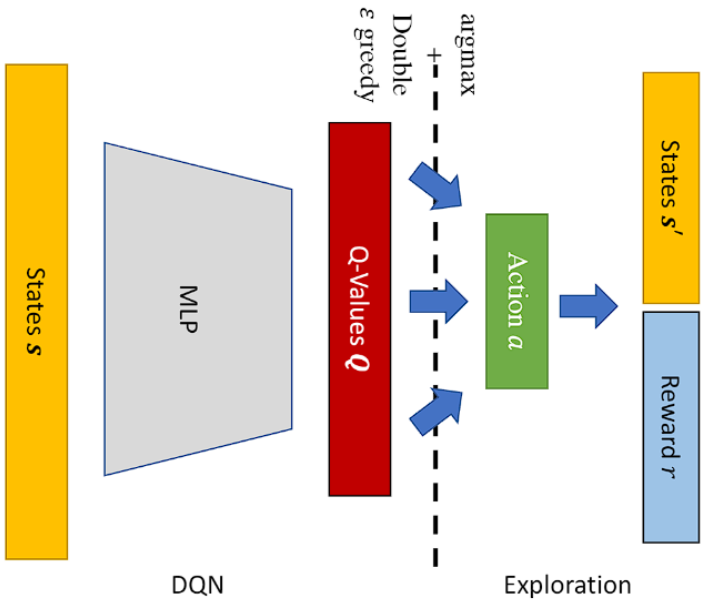

Figure 1.1: The whole pipeline in one decision epoch. Taking an old state s and transiting to the next state s0 with a feedback r [4].

1.2 Wireless Edge Caching

Data traffic is ever-increasing, straining networks and slowing down user experiences. Wireless caching emerges as a powerful technique to address this challenge. It works by strategically storing popular content at various points within a wireless network, closer to the users who are likely to request it.

By fulfilling user requests from cached content at base stations or access points, the network no longer needs to rely solely on the central source for everything. This translates to faster download speeds and smoother streaming, especially during periods of peak traffic. Lower latency, or delay, is another crucial advantage. When users access cached content, they experience faster response times and reduced buffering, leading to a more enjoyable experience for applications like video calls and online gaming. Furthermore, the ability to cache content at the network edge, on access points or even user devices, allows for offline access in situations with limited or no connectivity. This is particularly beneficial for areas with poor internet infrastructure or for frequently used content that doesn't require real-time updates.

However, wireless caching also presents its own set of challenges. One critical aspect is cache management. Deciding what content to cache, where to store it, and how to

handle updates requires careful planning. Caching less popular content wastes valuable storage space on devices at the network edge, which typically have less capacity compared to centralized servers. On the other hand, neglecting to cache frequently accessed data defeats the purpose of the system altogether. Additionally, keeping cached content up-to-date can be complex, especially for dynamic content that is constantly changing. Another challenge is predicting content popularity. Accurately forecasting which content will be in high demand is no easy feat. User preferences can be dynamic and influenced by various factors, making it difficult to pinpoint exactly what will be popular.

By strategically storing popular content at the network edge, it tackles congestion, reduces latency, and enables offline access. However, effectively managing cache resources, content selection, and updates remains an ongoing area of research to fully harness the potential of this technology.

CHAPTER 2

Literature Survey

2.1 DRL in Caching

Traditional cache replacement policies often fall short in dynamic environments like the variable nature of user requests that introduces temporal fluctuations, necessitating adaptable solutions. Base stations, constrained by finite cache capacities, typically rely on cache replacement policies like Least Recently Used (LRU) or First-In-First-Out (FIFO) to manage content [3], However, these policies struggle to accurately handle real-world complexities, particularly dynamic content popularity patterns Recent initiatives draw inspiration from reinforcement learning (RL) and its proven effectiveness in addressing complex control challenges. These endeavors harness deep neural networks (DNNs) [4] in conjunction with model-free deep RL (DRL) to optimize long-term rewards in mobile edge caching [2]-[4]. Edge nodes proactively retrieve content from source servers to replace local cache content during cache misses [5].However, the fetch-and-replace strategy has limitations. Freshly fetched content might not consistently surpass cached content in popularity. This underscores the need for a cache management approach that considers the temporal dynamics of user requests and content popularity. By integrating insights from reinforcement learning, deep neural networks, and content popularity prediction, the goal is to develop a cache management framework that optimizes cache resources while adapting to evolving wireless content consumption.

2.2 Coded caching

Coded caching, is the idea of using carefully designed user cache content to enable content delivery via coded multicast transmissions. First introduced by Maddah-Ali and Niesen in [6] coded caching has since been the subject of a large number of studies seeking to extend the original scheme into more practical scenarios. The coded caching scheme of [6] is developed for a system in which a central server has complete knowledge of user numbers and identities, users have identical cache sizes and make a single download request, transmission occurs over an error free link, and files are of equal length and popularity. Subsequent work has since extended the coded caching idea to the decentralized system, and to systems with non uniform file popularity , non-uniform file length multiple user requests , non-uniform cache size , and non-uniform channel quality [7]. The setup studied in [6] and [8] consisted of single-level content, i.e., every file in the system is uniformly demanded.However, it is well understood that content demand is nonuniform in practice, with some files being more popular than others [9]. Therefore we propose a scheme that considers non-uniform file popularity in order to achieve better transmission rates.

CHAPTER 3

Project Overview

3.1 Problem Statement

As data traffic continues to surge and technologies like Internet of Things (Iot) and 5G become more prevalent, the demand for quick data delivery is on the rise. Caching systems are pivotal in facilitating speedy access to frequently requested content. Hence, it's essential to manage caches effectively to cope with the increasing number of user requests. By conducting a comprehensive comparison of various existing caching algorithms, our objective is to develop a more adaptive and efficient caching methodology. The strategy aims to optimize cache hit rates, thereby ensuring improved performance and reliability in cache management systems.

3.2 Background

3.2.1 Deep Reinforcement Learning:

Deep Reinforcement Learning (DRL) is an emerging field in machine learning that merges two powerful techniques: reinforcement learning (RL) and deep learning. RL focuses on training agents to make optimal decisions through trial and error in an interactive environment. The goal is to maximize rewards received for each action taken. Deep learning, on the other hand, excels at uncovering intricate relationships between inputs and outputs using artificial neural networks. These networks can learn complex features directly from raw data.

DRL leverages deep neural networks to represent an RL agent's policy or value function. This network is trained using RL techniques like Q-learning and policy gradients. It analyzes the current state of the environment and recommends an action. The action is then executed, and the agent receives a reward signal that fine-tunes the network's settings. Deep neural networks are particularly adept at learning intricate state-action spaces, allowing agents to excel in tasks with numerous possible actions or states.

DRL algorithms frequently employ a neural network as a function approximator to learn the optimal policy or value function for the agent. The network takes in data about the environment's current state and suggests an action for the agent. The agent executes the action, and the environment responds with a reward signal that is used to update the network's parameters. The network architecture plays a crucial role in the effectiveness of DRL algorithms. To capture spatial and temporal relationships in the input data, deep neural networks often incorporate convolutional and recurrent layers. The specific task and the environment's characteristics determine the network's topology and the RL algorithm used for updating its parameters.

3.2.2 Coded Caching

Coded caching is a strategy for reducing the amount of data that must be transferred in distributed systems. It operates by pre-caching certain data in the receiver's cache. As a result, the overall cost of communication can be greatly lowered.

Coded caching achieve a significant reduction in network load by creating and exploiting coded multicasting opportunities between users with different demands[madali]. Core Functionalities of coded caching includes,

- **Collaborative Content Storage:** Unlike traditional caching where users store individual content items, coded caching employs a strategic approach. Users cache a subset of content from a vast library stored on the server. This selection can be based on user preferences, content popularity, or a caching algorithm designed to maximize the benefit of coded transmissions.

- **Coded Multicast Transmission:** A fundamental innovation in coded caching lies in the content delivery mechanism. Unlike traditional unicasting, where individual data packets are transmitted directly to each user, coded caching employs a technique called coded multicast transmission. In this approach, the server transmits strategically encoded data packets termed codewords. These codewords encapsulate fragments of information relevant to the content requests of multiple users. This strategic encoding enables users to leverage the received codewords in conjunction with their cached content to reconstruct the desired data, even if it wasn't explicitly stored in their individual caches.

- **Cooperative Decoding:** Upon receiving the codewords, users don't simply accept them as complete data. They leverage a powerful tool – decoding algorithms. These algorithms exploit the relationships between the cached content on the user device and the information received in the codewords. By combining these two sources of information, users can reconstruct the complete files they require, even if they haven't explicitly cached them.

Traditionally, content delivery networks (CDNs) and caching servers rely on uncoded caching, where users store a portion of the content library locally on their devices.

This approach offers some benefits in terms of reducing network traffic and improving user experience for frequently accessed content. However, it has limitations, Limited Efficiency and High Server Load. Coded caching offers a more sophisticated approach that addresses these limitations and coded caching achieves a much lower delivery rate compared to uncoded caching, leading to a more efficient content delivery system.

Rate for uncoded caching:

$$R_{uncoded} = K\left(1 - \frac{M}{N}\right) \tag{3.1}$$

Rate for coded caching:

$$R_{coded} = K\left(1 - \frac{M}{N}\right)\left(\frac{1}{1 + \frac{KM}{N}}\right) \tag{3.2}$$

Figure 2.1 compares coded and uncoded caching approaches, focusing on the relationship between cache size and the number of transmissions required. In uncoded caching, there is a gradual decrease in transmissions as cache size increases, showing a linear correlation. In contrast, coded caching demonstrates a significant reduction in transmission counts as cache size increases, indicating an exponential relationship. This highlights the superior effectiveness of coded caching in utilizing cache memory to reduce bandwidth requirements.

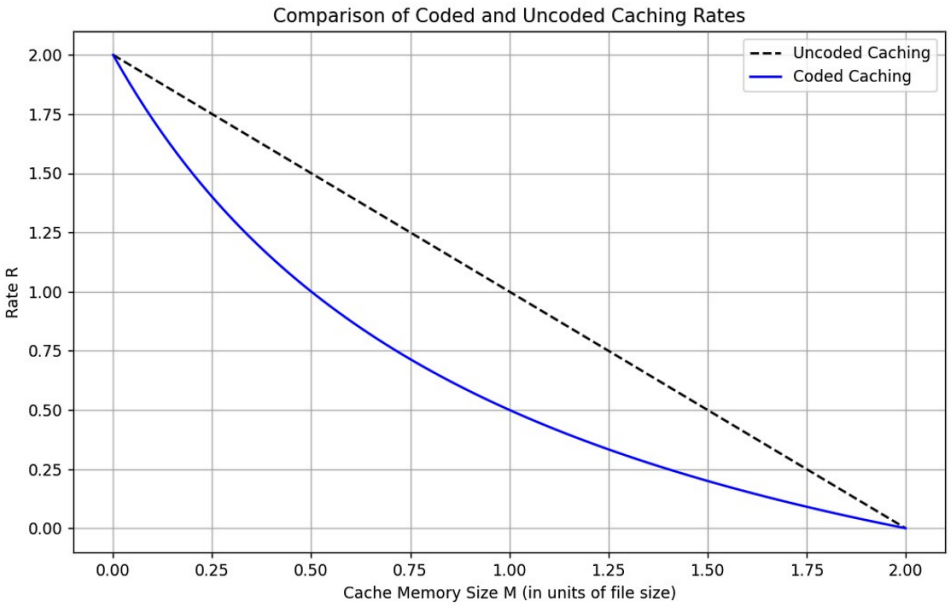

Figure 3.1: Comparison between coded and uncoded caching

3.3 Objective

- **Dynamic Cache Management with Deep Q-Learning (DQN):** This objective tackles the challenge of inflexible cache replacement policies. We propose implementing a DQN agent trained through reinforcement learning. This will intelligently select the most valuable content to cache, considering popularity and user access patterns. It will also determine optimal placement within the network and choose the least valuable data to replace when storage fills up. This approach would lead to a more dynamic and efficient caching system.

- **Enhanced Hit Rates with Popularity-Aware Coded Caching:**Traditional caching struggles with large or non-redundant content. We propose exploring coded caching, where data fragments are spread across caches. This allows users to retrieve complete content even if they only access a subset. Our focus will be on implementing a scheme that considers content popularity. This means allocating more fragments of popular content across caches, increasing the likelihood of users finding what they need quickly. This would significantly improve cache hit rates, especially for larger content pieces.

3.4 Motivation

- **Adaptive Caching Strategies with Deep Reinforcement Learning (DRL):** Traditional caching schemes like LRU, LFU, and FIFO rely on fixed rules that might not adapt well to real-world access patterns. Deep reinforcement learning (DRL) offers a more dynamic solution. By learning from user requests and past access history, DRL can make informed caching decisions based on a wider range of factors, including content size, popularity trends, and even user preferences. This adaptability is particularly valuable in caching, where DRL can significantly improve cache hit rates and user experience compared to traditional, static approaches.

- **Reduced Network Strain with Coded Caching:** Coded caching shines compared to uncoded caching by reducing network strain during peak usage. Unlike uncoded caching where entire files sit in user caches, coded caching breaks them down and distributes pieces strategically. The server then transmits a single message that multiple users can use to reconstruct the file from their combined cache pieces, significantly reducing overall data transmission. This approach leverages user cache space as a whole, even if no single user has the complete file. Coded caching scales well with more users as the collective chance of having necessary pieces for reconstruction increases. However, implementing it can be more complex and might not be ideal for situations with highly diverse user demands.

CHAPTER 4

Proposed Plan

4.1 Part 1:

4.1.1 System Model

A basic wireless caching network comprising a source server and a cache-enabled base station (BS) is considered. The BS connects to the source server via a wireless backhaul [1]. The set $O = \{o1, o2, ..., o|O|\}$ represents all $|O|$ contents available in the server. Due to storage limitations, the BS can predownload a maximum of N contents from the server. It's important to note that the contents in the server encompass all potential requests from all users in real time. The focus of this research is on a caching scenario where only a small fraction of the server's contents are requested and subsequently prefetched by the BS. In other words, $|O| >> N$. Despite the BS's limited cache storage, each user can request up to N contents. It's also crucial to consider that the BS can receive these requests from multiple users without having prior knowledge of content popularities. To efficiently address the ever-changing requests over time, the BS needs to update its local cache accordingly.

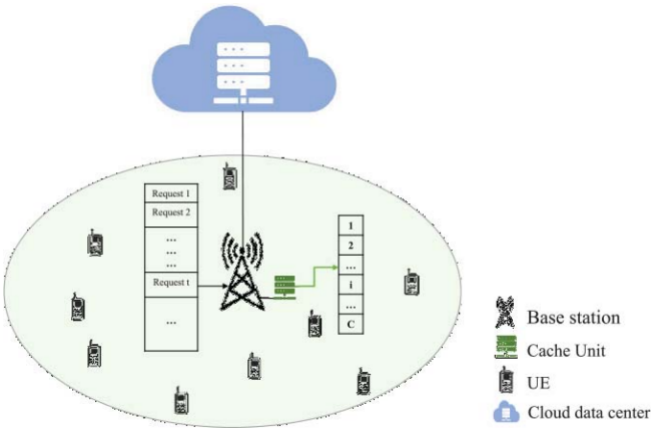

Figure 4.1: System model of a centralized caching system.

4.1.2 Implementation details

Feature Extraction

Each state is encoded by $s_t = \{F_0, F_1, ..., F_C\}$, where F_0 concatenates the features of cached blocks and the currently requested block. Here, $F_i = \{f_{is}, f_{im}, f_{il}\}$, representing short-term, middle-term, and long-term recent access times. Frequency information is employed due to its adaptability observed in LFU and the temporal nature of data access. However,to mitigate high memory costs of storing all-time access times, only a small subset of history is kept and a neural network is used to regress and explore latent information.Past 10, 100, and 1000 accesses were considered.

Reward

For the reward function values were set as $\mu = 10$, $\psi = 50$, $\kappa = 1$, and $\lambda = 1.0$ in the penalty term. The penalty term often makes less difference but can significantly assist DRL in extreme cases. When reaching the end state, i.e., all accesses are handled, the reward still follows the formulation of Equation 1 but without the penalty term.

DQN Hyperparameters

The basic learning parameters are set as learning rate $\alpha = 0.01$ and batch size $N_\alpha = 32$. The MLP consists of one hidden layer with 256 perceptrons. The network is not deepened, as a small-scale network is easier to converge. Given the dynamic nature of the cache system world, quick convergence is essential. The memory list has a size of $N_\beta = 10000$, and the network is trained every $T_1 = 5$ transitions. The parameters synchronize every $T_0 = 100$ decision epochs. As for the parameters of Q-Learning with ϵ-greedy, discount factor is set as $\gamma = 0.9$. Exploration-exploitation factors $\epsilon 1$, $\epsilon 2$ are initialized with 0.1, 0.5, respectively. For $\epsilon 1$, the adjustment values are $\delta+ = 0.005$ and $\delta\text{-} = 0.005$. For $\epsilon 2$, $\delta+ = 0.1$ and $\delta\text{-} = 0.001$. A maximum value of 0.5 is set for every $\in$. The threshold ρ ranges from 5 to 50, and this value is tuned the experiment to acquire better results.

Data Generation and Simulation

The dataset is composed of two distinct parts, each serving a specific role. In the simulation phase, a probabilistic distribution needs to be established for requested content IDs. To achieve this, the Zipf distribution is employed, a widely used model for content popularity in networking simulations. The distribution parameter is set to 1.3, which allows to generate 10,000 requests to blocks in the file system, ensuring an ample number of cache misses for the experimental setup.

The second part of the dataset is obtained through real data collection in the Pintos operating system. By integrating a data collection process into the Pintos kernel, every accessed block ID is recorded in the output file. This real-world data is derived from six typical user programs selected from the Pintos test cases, providing a controlled yet diverse environment. With over 10,000 requests in these real test cases, the learning agent has the opportunity to observe and learn from authentic scenarios, enhancing the practicality of the experimental setup.

4.2 Drawbacks Identified

Upon thorough analysis of our previous results, we uncovered certain drawbacks that necessitated a reevaluation of our approach:

Limitations in System Model

Our system model, which featured a centralized cache at the base station alongside a server containing all files, was initially designed to accommodate a single user generating requests. However, upon reflection, we realized that this model may not effectively scale to multiple users. This limitation arises from the fact that, regardless of the number of users, a set of requests would converge at the base station, resembling the scenario of a single user. Additionally, we noted that we didn't fully utilize the potential of the cache storage available in user devices to improve cache hit rates.

Insufficient Improvement in Cache Miss Rates

Analysis of the episodic rewards generated by the Deep Reinforcement Learning (DRL) algorithm revealed a lack of significant improvement in cache miss rates over successive episodes. Furthermore, despite the high computational complexity associated with employing a deep reinforcement learning model, the observed improvements in cache miss rates were minimal when compared to the Least Frequently Used (LFU) algorithm, highlighting the need for more effective caching strategies.

Figure 4 shows episodic rewards generated by the Deep Reinforcement Learning (DRL) algorithm. It reveals a lack of significant improvement in cache miss rates over successive episodes

```
Agent=DQN, Size=5, Episode=0: Accesses=10000, Hits=4629, MissRate=0.462900
Agent=DQN, Size=5, Episode=1: Accesses=10000, Hits=4649, MissRate=0.464900
Agent=DQN, Size=5, Episode=2: Accesses=10000, Hits=4691, MissRate=0.469100
Agent=DQN, Size=5, Episode=3: Accesses=10000, Hits=4645, MissRate=0.464500
Agent=DQN, Size=5, Episode=4: Accesses=10000, Hits=4665, MissRate=0.466500
Agent=DQN, Size=5, Episode=5: Accesses=10000, Hits=4658, MissRate=0.465800
Agent=DQN, Size=5, Episode=6: Accesses=10000, Hits=4660, MissRate=0.466000
Agent=DQN, Size=5, Episode=7: Accesses=10000, Hits=4669, MissRate=0.466900
Agent=DQN, Size=5, Episode=8: Accesses=10000, Hits=4651, MissRate=0.465100
Agent=DQN, Size=5, Episode=9: Accesses=10000, Hits=4637, MissRate=0.463700
Agent=DQN, Size=5, Episode=10: Accesses=10000, Hits=4633, MissRate=0.463300
```

Figure 4.2: Analysis of episodic miss rates

4.3 Revised Approach

Based on our analysis and identified drawbacks, we decided to revise our approach for the current semester. This involved two main strategies:

1. Transitioning from a centralized cache architecture to a distributed model, where each user would have an individual cache.

2. Exploring the potential of coded caching techniques to further enhance cache hit rates and optimize content delivery efficiency. This decision was motivated by the aim to leverage advanced coding methods to maximize cache utilization and improve overall system performance.

By implementing both a distributed caching model and coded caching techniques, we intend to compare their efficiency with that of the Deep Reinforcement Learning (DRL) algorithm. This comparative analysis will provide valuable insights into the relative effectiveness of these approaches.

4.4 Part 2:

4.4.1 System Model

The system model depicted in Figure 2 illustrates a centralized caching architecture, where a server, equipped with a repository of N distinct files each of size F bits, is interconnected with K users via a shared communication link. Each user is provisioned with a local cache of size M F bits, enabling the storage of content proportional to the size of an individual file. This configuration is designed to exploit the spatial redundancy of user demands by caching content closer to the end-users., thereby alleviating the load on the shared network link and reducing the file retrieval latency.

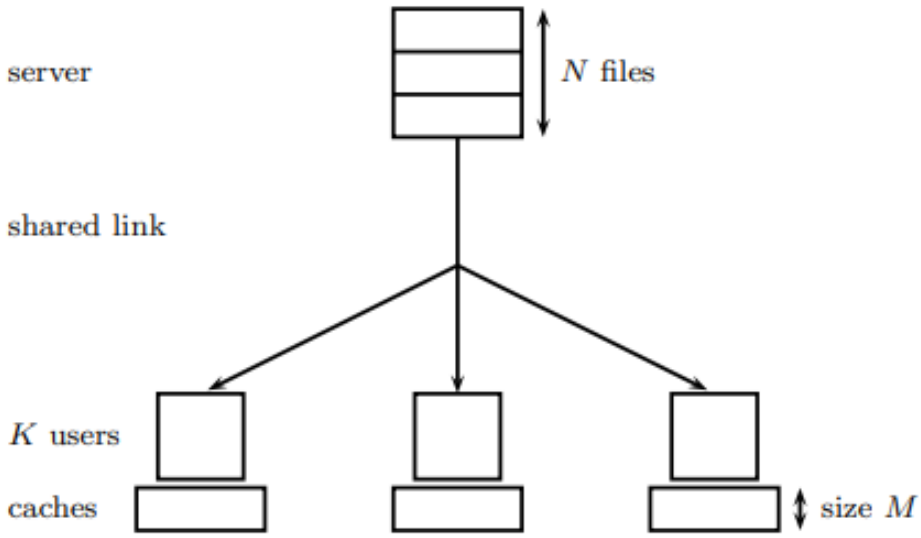

Figure 4.3: System model for coded caching

4.4.2 Placement and Delivery procedures

This section proposes a coded caching scheme to improve cache hit rates and reduce transmission overhead in a wireless network with N files, each of size F bits. The network caters to K users, each equipped with an isolated cache memory of size M bits. The scheme uses the concept of coded caching, where each file is fragmented into smaller subparts. These subparts are strategically distributed across user caches to enable efficient content retrieval. Here's a breakdown of the key aspects:

1.Popularity-Based Fragment Allocation

- We denote the probability of a request for file i as p_i.
- During the placement phase, each file is divided into $2N$ subparts.
- $n_i = \frac{p_i \cdot 2N}{\sum_{i=1}^{N} p_i}$ (where p_i can be obtained by rounding $(100 \cdot p_i)$ to an integer) (Eq. 1)
- Equation 1 (Eq. 1) gives the number of subparts of the ith file that will be stored in the cache. This allocation considers the file's access popularity (p_i) to ensure a higher number of subparts are stored for frequently requested files.

2.Cache Capacity Constraints

- The value of n_i is rounded to the nearest integer to ensure efficient storage utilization.
- The sum of n_i across all files cannot exceed the total number of available subparts, $2N$, to avoid exceeding cache capacity. This constraint ensures that the cache allocation remains feasible within its limitations.

3.Delivery Phase

- When multiple users request files, the server leverages XOR operations for efficient transmission. If a requested file's subparts are scattered across different user caches, the server transmits the XOR of those subparts. This eliminates redundant data transmission in overlapping subparts, significantly reducing the overall transmission burden.

- Each user can then decode the received XOR data using the subparts stored locally in their cache. This distributed decoding approach substantially reduces the amount of data transmitted from the server compared to sending complete files, leading to improved transmission efficiency.

Example 1

Consider a scenario with three files (A, B, and C), two users ($p_K = 2$), and a cache size of one unit ($p_M = 1$). The access probabilities for files A, B, and C are denoted as $p_A = 0.5$, $p_B = 0.3$, and $p_C = 0.2$, respectively.

Following Eq.1, the fragment allocation would be $n_A = 3$, $n_B = 2$, and $n_C = 1$, reflecting the higher popularity of file A. User caches would then be populated with distinct subparts for the same file, maximizing the chance of finding requested content. Therefore, user 1's cache has files [A1, A2, A3, B1, B2, C1] and user 2's cache has files [A4, A5, A6, B3, B4, C2].

Based on the cache content of User 1 and User 2, here are some delivery scenarios using XOR operations:

- Case 1: Request1 is p_A and Request2 is p_A
  - Send → A1 xor A4, A2 xor A5, A3 xor A6
- Case 2: Request1 is p_A and Request2 is p_B
  - Send → A4 xor B1, A5 xor B2, A6, B5, B6
- Case 3: Request1 is p_B and Request2 is p_B
  - Send → B1 xor B3, B2 xor B4, B5, B6

- Case 4: Request1 is p_C and Request2 is p_A
  - Send $\rightarrow$ A1 xor C2, C3, C4, C5, C6, A2, A3
- Case 5: Request1 is p_C and Request2 is p_B
  - Send $\rightarrow$ C2 xor B1, C3, C4, C5, C6, B2, B5, B6
- Case 6: Request1 is p_C and Request2 is p_C
  - Send $\rightarrow$ C1 xor C2, C3, C4, C5, C6

The table shown below calculates the average transmission rate for the example discussed above.

Table 4.1: Transmission Rate for $N = 3$

User Request A	**User Request B**	**Average Rate(R)**	**Probablity (Pr)**	**R*Pr**
A	A	0.5	0.36	0.18
A	B	0.83	0.18	0.1494
A	C	1.17	0.06	0.0702
B	A	0.83	0.18	0.1494
B	B	0.67	0.09	0.0603
B	C	1.33	0.03	0.0399
C	A	1.17	0.06	0.0702
C	B	1.33	0.03	0.0399
C	C	0.83	0.01	0.0083

Example 2

Consider a scenario with four files (A, B, C, D), two users (K=2), and a cache size of one unit (M = 1). The access probabilities for files A, B, C and D are denoted as $p_A = 0.6$, $p_B = 0.2$, $p_C = 0.18$, and $p_D = 0.02$, respectively.
Following Eq. 1, the fragment allocation would be $n_A = 5$, $n_B = 2$, $n_C = 1$, and $n_D = 0$, reflecting the higher popularity of file A. Therefore, user 1's cache has files [A1, A2, A3, A4, A5, B1, B2, C1] and user 2's cache has files [A4, A5, A6, A7, A8, B3, B4, C2]. The delivery can be done as follows:

- Case 1: Request1 is A and Request2 is A
  - Send → A6 xor A1, A7 xor A2, A8 xor A3
- Case 2: Request1 is A and Request2 is B
  - Send → A6 xor B1, A7 xor B2, A8, B3, B4, B5, B6
- Case 3: Request1 is A and Request2 is C
  - Send → A6 xor C1, A7, A8, C2, C3, C4, C5, C6
- Case 4: Request1 is A and Request2 is D
  - Send → A6, A7, A8, D1, D2, D3, D4, D5, D6, D7, D8
- Case 5: Request1 is B and Request2 is B
  - Send → B1 xor B3, B2 xor B4, B5, B6, B7, B8
- Case 6: Request1 is B and Request2 is C
  - Send → B3 xor C1, B4, B5, B6, B7, B8, C3, C4, C5, C6, C7, C8
- Case 7: Request1 is B and Request2 is D
  - Send → B3, B4, B5, B6, B7, B8, D1, . . . , D8
- Case 8: Request1 is C and Request2 is C
  - Send → C1 xor C2, C3, C4, C5, C6, C7, C8
- Case 9: Request1 is C and Request2 is D
  - Send → C2, C3, C4, C5, C6, C7, C8, D1, . . . , D8
- Case 10: Request1 is D and Request2 is D
  - Send → D1, D2, D3, D4, D5, D6, D7, D8

The table shown below calculates the average transmission rate for the example discussed above.

Table 4.2: Transmission Rate for $N = 4$

User Request A	**User Request B**	**Average Rate(R)**	**Probablity (Pr)**	**R*Pr**
A	A	0.375	0.36	0.135
A	B	0.875	0.12	0.105
A	C	1.125	0.108	0.1215
A	D	1.375	0.012	0.0165
B	B	0.75	0.04	0.03
B	C	1.5	0.036	0.054
B	D	1.75	0.006	0.0105
C	C	0.875	0.0324	0.02835
C	D	1.875	0.0036	0.00675
D	D	1	0.0004	0.0004

The average transmission rate, denoted by $R(M, N, p)$, can be expressed using the following equation:

$$R(M,N,p) = \frac{\sum_{i=1}^{N}(2N - M \cdot n_i) \cdot p_i^2}{2N} + \frac{2 \cdot \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (4N - 2M \cdot n_j - M \cdot n_i) \cdot p_i \cdot p_j}{2N} \tag{4.1}$$

CHAPTER 5
Results and Discussion

5.0.1 Enhancing edge caching using DRL

The simulation results we obtained indicated promising outcomes, with our approach demonstrating reduced cache miss rates compared to traditional methods like LRU. However, upon analysis, we identified limitations in the previous work, particularly in the adaptability of caching strategies to varying user demands(discussed in section 3.2). Figure 5.1 shows the simulation results of the cache miss rate of the DQN algorithm compared to other algorithms.

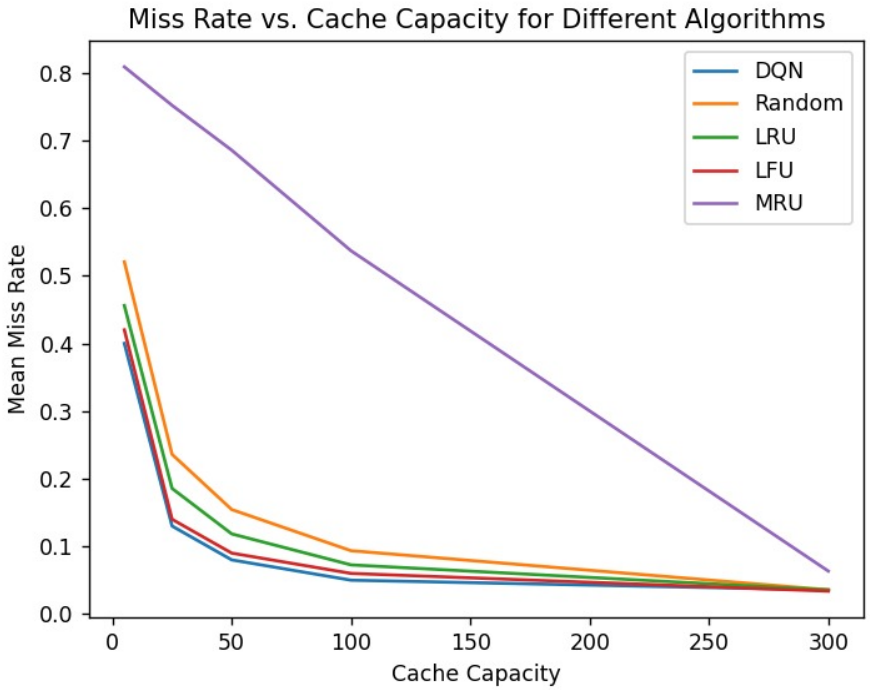

Figure 5.1: Cache miss rates of different algorithms

5.0.2 Coded Caching with Non-Uniform Content Popularity

Figure 5.2 illustrates how the transmission rate changes for the two coded caching schemes as the popularity of a single file increases. The graph shows that Maddah-Ali's scheme performs better when the access probabilities of the files are more similar. This suggests that the number of fragments assigned to each file during placement needs to be adapted based on its individual popularity. In simpler terms, the caching strategy should prioritize storing more fragments of files that are more likely to be requested by users.

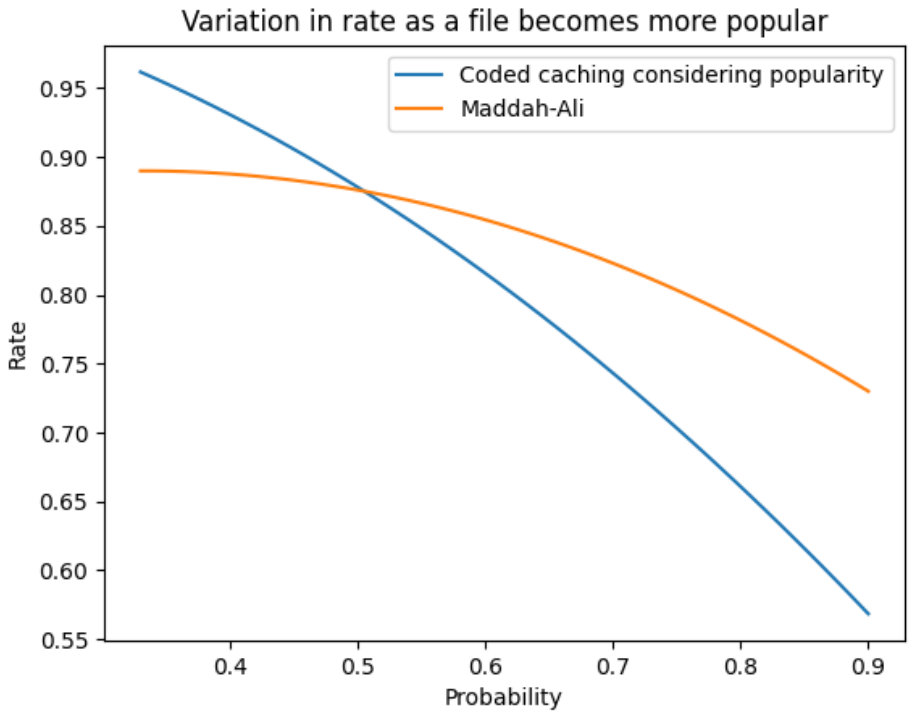

Figure 5.2: Variation in rate as a file becomes more popular

Figure 5.3 demonstrates the impact of cache size on transmission rate for the two coded caching schemes. The scheme that considers non-uniform content popularity achieves a better transmission rate compared to the scheme that assumes uniform popularity. This signifies that allocating cache space based on a file's access probability leads to a more efficient use of resources and ultimately reduces the amount of data that needs to be transmitted.

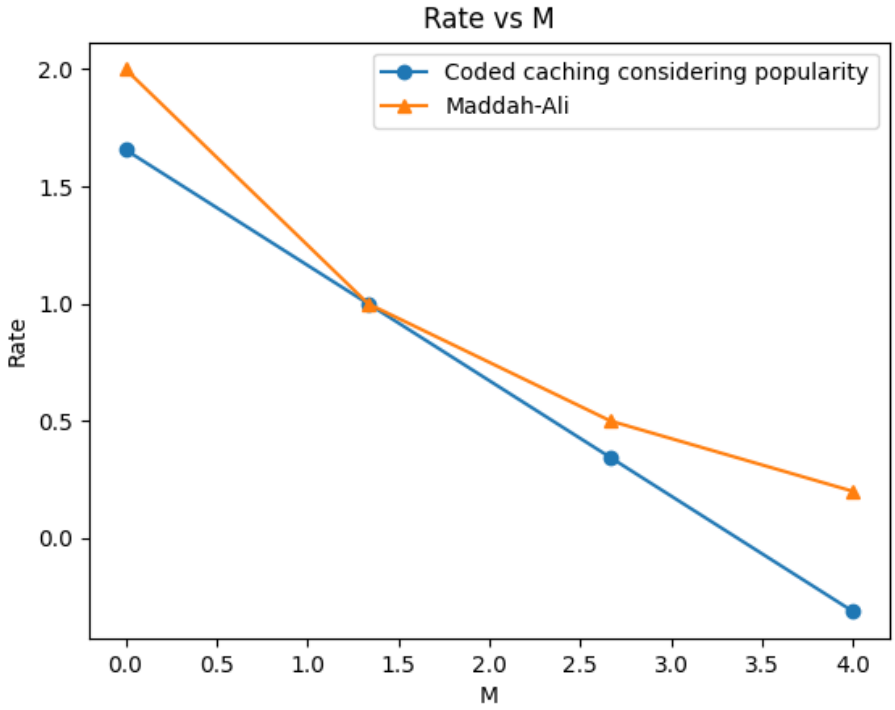

Figure 5.3: Rate vs M

CHAPTER 6

Conclusion

This project investigated the potential of coded caching to improve cache hit rates and reduce transmission overhead in wireless networks. We began by implementing a cache replacement strategy based on reinforcement learning. This approach, while achieving significant improvement in cache hit rates compared to traditional policies, faced limitations. The single-cache model at the base station did not fully utilize user device caches, and miss rates did not show consistent improvement. Therefore, we shifted our focus to exploring coded caching. We identified that existing schemes often assume uniform content popularity. To address this, we implemented a coded caching scheme that considers non-uniform popularity by allocating cache fragments based on a file's access probability. This popularity-aware approach improved the transmission efficiency compared to the uniform-popularity scheme.

In conclusion, this project highlights the potential of coded caching for enhancing wireless network performance. By incorporating user device caches and considering non-uniform content popularity, coded caching offers a promising approach to improve cache hit rates and reduce transmission overhead, leading to a more efficient and user-centric wireless network experience.

BIBLIOGRAPHY

[1] P. Wu, J. Li, L. Shi, M. Ding, K. Cai and F. Yang, "Dynamic Content Update for Wireless Edge Caching via Deep Reinforcement Learning," in IEEE Communications Letters, vol. 23, no. 10, pp. 1773-1777, Oct. 2019, doi: 10.1109/LCOMM.2019.2931688.

[2] C. Zhong, M. C. Gursoy and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," 2018 52nd Annual Conference on Information Sciences and Systems (CISS), Princeton, NJ, USA, 2018, pp. 1-6, doi: 10.1109/CISS.2018.8362276.

[3] H. Zhu, Y. Cao, W. Wang, T. Jiang and S. Jin, "Deep Reinforcement Learning for Mobile Edge Caching: Review, New Features, and Open Issues," in IEEE Network, vol. 32, no. 6, pp. 50-57, November/December 2018, doi: 10.1109/MNET.2018.1800109.

[4] H. Pang, J. Liu, X. Fan and L. Sun, "Toward Smart and Cooperative Edge Caching for 5G Networks: A Deep Learning Based Approach," 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 2018, pp. 1-6, doi: 10.1109/IWQoS.2018.8624176.

[5] X. Wu, J. Li, M. Xiao, P. C. Ching and H. V. Poor, "Multi-Agent Reinforcement Learning for Cooperative Coded Caching via Homotopy Optimization," in IEEE Transactions on Wireless Communications, vol. 20, no. 8, pp. 5258-5272, Aug. 2021, doi: 10.1109/TWC.2021.3066458.

[6] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, 2013, pp. 1077-1081, doi: 10.1109/ISIT.2013.6620392.

[7] A. M. Daniel and W. Yu, "Optimization of Heterogeneous Coded Caching," in IEEE Transactions on Information Theory, vol. 66, no. 3, pp. 1893-1919, March 2020, doi: 10.1109/TIT.2019.2962495.

[8] M. A. Maddah-Ali and Urs Niesen, “Decentralized coded caching attains order-optimal memory-rate tradeoff,” IEEE/ACM Trans. Netw., vol. 23,no. 4, pp. 1029–1040, Aug. 2015.

[9] [J. Hachem, N. Karamchandani and S. N. Diggavi, "Coded Caching for Multi-level Popularity and Access," in IEEE Transactions on Information Theory, vol. 63, no. 5, pp. 3108-3141, May 2017, doi: 10.1109/TIT.2017.2664817.]