CMPE 49F FINAL PROJECT REPORT

Mustafa Enes Çakır, Sinem Kafiloglu and Fatih Alagöz
Department of Computer Engineering, Bogazici University
Istanbul, Turkey
Email: enes.cakir, sinem.kafiloglu, fatih.alagoz@boun.edu.tr

TABLE I
DEFAULT SYSTEM PARAMETERS

Parameter	Explanation	Value
N_c	The total number of contents	100
s	The Zipf distribution skewness parameter	0.8
$Cache_{dev}$	The device cache capacity	1 Gbits
λ_{base}	The mean of base content size	25 Mbits
λ_{enh}	The mean of enhancement content size	5 Mbits
p_{HQ}	The rate of the SU requests high quality	0.5
λ_{user}	The mean of PPP density of users	$0.0015\frac{user}{m^2}$
R	The radius of D2D region	300 m
R_d	The radius of device lookup region	146 m
λ_{PU}	The mean PU arrival	$1\frac{user}{sec}$
λ_{SU}	The mean SU arrival	$1.5\frac{user}{sec}$
N_{ch}	The total number of channels	10
f_1	The initial frequency	700 MHz
N_0	The white gaussian noise	$1.6e^{-19}\frac{Watt}{Hz}$
t_{base}	The base local hit latency	0.25 sec
t_{enh}	The enhancement local hit latency	0.05 sec

I. INTRODUCTION

Every second, thousands of seconds video is uploaded to video services. The most of the Internet traffic is consumed by uploading and downloading these videos. Uploading is a one time process, but each new visitor downloads the same video again and again. Delivering contents to clients efficiently is a troublesome process. Classical networking solutions do not solve this problem smoothly. Caching content on devices that closer to clients is more convenient than centralized caching. This method raises another problem to think. Which devices are cache which contents? How we distribute cache to devices?

In our study, we tried to reduce network latency and increase local hit rate. We focused local cached distribution. Each device looks neighbor devices for what to replace.

The rest of this paper is organized as follows. The next section is about the model of our system and the assumptions that we made. In Section III, we describe the detail of our algorithm with psedocode and follow steps of it with sample input. Section IV shows the performance result of our simulations and compares our algorithm with LRU, LFU, and RANDOM. Finally, Section V sums up our paper and gives a summary of the algorithm.

II. SYSTEM MODEL

Our models have two different types of users: primary and secondary. Primary users are the real owner of the frequencies. Secondary users use frequencies when they are idle. Our caching algorithm is used by secondary users. We do not care about the internal processes of the primary users. We can not use the selected frequency while a primary user is using it. It is the only thing we care about primary users. A primary user cannot preempt the other primary user. It can only stop the service of the secondary users. We don't have any queue for `BLOCKED` and `DROPPED` users. When they are `BLOCKED` or `DROPPED`, they leave the system.

Secondary users distribute with Poisson Point Process into the region with the radius R. When a new secondary user arrives the system, it requests a new content depends on their popularity. By the way, we have N distinct contents and their popularity calculate with Zipf distribution. Each content has base and enhancement layer. User request high quality with the rate of p_{HQ}.

When a user request content, the user cache the content certainly. There is no option for not caching content. Each device has $cache_{dev}$ cache storage capacity. So they cannot store all content versions. Our algorithm serves here. It helps to decide which cache content to replace. If the cache has new content, it means local hit. The user doesn't need a frequency channel. If local cache has not new content, user look for content to neighbors. If wanted content is found in another device, the secondary user request an idle channel. If it found idle one, starts to transfer video. If it couldn't find an idle frequency, the request is blocked.

If any other device has not the requested content, also request is blocked. We don't have a base station. We have N_{ch} different frequencies at our D2D network.

III. Our Content Caching Algorithm

The main goal of our caching algorithm is that distribute contents to local areas with weighted by the count in the area and its popularity. The local area is a region with R_d. R_d is calculated based on the idea that each user should look nearest N other users. So we can calculate R_d as below.

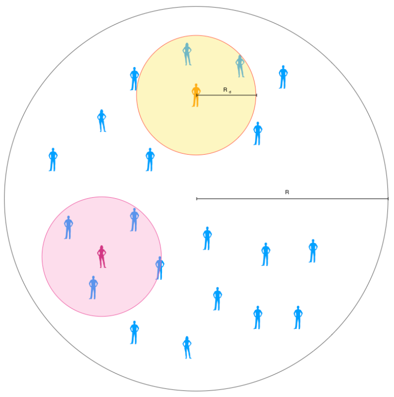

Fig. 1. The diameter of device, R_d

$$R_d = \sqrt{\frac{N/\lambda_{user}}{\pi}}$$

The λ_{base} is 5 times of λ_{enh}. In average, we should delete 5 enhancement layers for a new base layer. It is not efficient. We replace enhancement with enhancement, vice verse for the base.

Our algorithm calculates the weight of each content by dividing the count of the content in neighbor device's caches by the popularity of content.

$$W_k = \frac{count_k}{popularity_k}$$

The content with the biggest weight is replaced because we have enough count of this content in this area.

If the device cache doesn't have the same type (base or enhancement) of the layer, it replaces with the other type.

Algorithm 1: Our cache replacement algorithm

```
Input: New cache to store: $new\_cache$, the
       current secondary user: $user$
Output: The cached content to remove
1  $cache\_counts \leftarrow a\ new\ counter$
2  $neighbors \leftarrow users$ where $distance \leq R_d$
   for $neighbor$ in $neighbors$ do
3      $cache\_counts \mathrel{+}= neighbor.caches_{counts}$
4  for $cache_i$ in $user.caches$ do
5      $weight_i \leftarrow cache\_counts_i\ /\ popularity_i$
6  $cache\_lookup \leftarrow user.caches$ where
     $cache_{type} == new\_cache_{type}$
7  if $cache\_lookup$ is empty then
8      $cache\_lookup \leftarrow user.caches$
9  Sort $cache\_lookup$ by weight in descending order
10 return First element of $cache\_lookup$
```

Example 1: *A base layer content is requested and the user has enough base layers on its cache storage.*

- The user's device gets cached contents identifiers from neighbor devices that closer than R_d.
- It counts each different type of content.
- It calculates the weight of each content by dividing their counts by their popularities.
- It filters the base content layers because it doesn't need to enhancement weights.
- It removes the filtered contents (only base layer on this example) that have the highest weight until we have enough space at its cache storage.

Example 2: *A enhancement layer content is requested and the user doesn't have enhancement layers on its cache storage.*

- The user's device gets cached contents identifiers from neighbor devices that closer than R_d.
- It counts each different type of content.
- It calculates the weight of each content by dividing their counts by their popularities.
- It doesn't filter the enhancement content layers, because it doesn't have an enhancement layer for replacement on cache storage.
- It removes the filtered contents (all layer content on this example) that have the highest weight until we have enough space at its cache storage.

IV. PERFORMANCE EVALUATION

We try to reduce latency while increasing local hit in our algorithm. We compare `LRU`, `LFU`, `RANDOM` and our algorithm with 5 different performance meter: $latency$, $p_{loc}^{SQ}(base)$, $p_{loc}^{HQ}(base)$, $p_{loc}^{HQ}(enh|base_{loc})$ and $p_{loc}^{HQ}(enh|base_{D2D})$.

First of all, we run 10 different simulations with $T_{sim} = 2000$ for each algorithm. But each time they give different performance ranking. For convergence results to some values, we run simulation 250 times with $T_{sim} = 5000$ for each algorithm. We reached more mature results with more simulations.

You can see channel usage history sample for different T_{sim} (100, 2000, 5000) next figures. We do not have heavy frequency usages. The majority of the channels are idle generally.

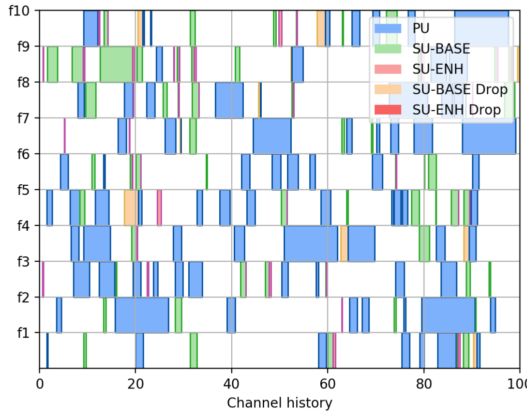

Fig. 2. Channel history example for $T_{sim} = 100$

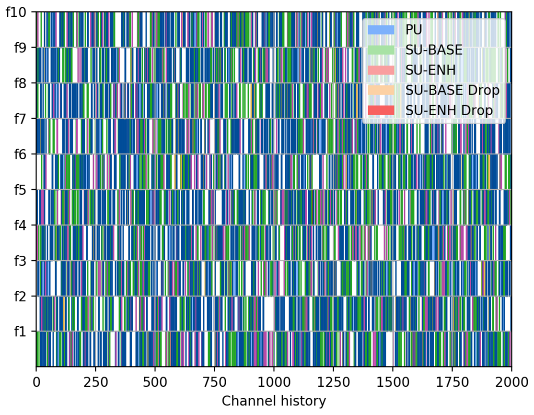

Fig. 3. Channel history example for $T_{sim} = 2000$

f10
f9
f8
f7
f6
f5
f4
f3
f2
f1
0
1000
2000
3000
4000
5000
Channel history
PU
SU-BASE
SU-ENH
SU-BASE Drop
SU-ENH Drop

Fig. 4. Channel history example for $T_{sim} = 5000$

The first performance criterion is the latency. The latency depends on the service time. In addition to that, the service time is the depends on the distance. Because when the distance is increased, Shannon's capacity is decreased. Our algorithm tries to decrease access distance to new content for each device. It increases the density of the content that sparse in the local area. So latency is decreased in our algorithm (see figure 5).

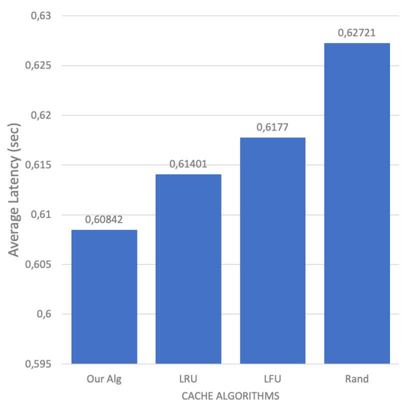

Fig. 5. The average $latency$ for algorithms

Secondary users choose new content with the probability that is their popularity. So they request more popular content frequently. Our algorithm distributes content density depends on their content and popularity. It puts popular items denser. Local hit rates are higher than *LRU*, *LFU*, *RANDOM*; because they do not take account into popularity.

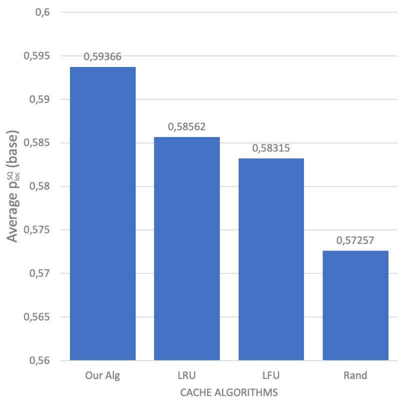

Fig. 6. The average $p_{loc}^{SQ}(base)$ for algorithms

Our device's cache storages are 1Gbits. Approximately, $60-70$ caches fit it. We have 100 different contents, so we have 200 layers for caching. We can cache $\%30$ of all contents. 1^{st} content has 15 times more chance than 30^{th} content because of Zipf distribution. So we cached content that probably will be the next requested one. $p_{loc}^{SQ}(base)$ is increased (see figure 6).

Generally, we have the same reasons for better local hit rates.

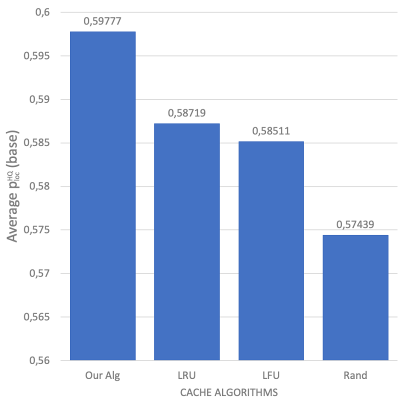

Fig. 7. The average $p_{loc}^{HQ}(base)$ for algorithms

Requesting SQ or HQ is depends on p_{HQ}. Request type doesn't depend on the content identifier. Still, the most popular ones selected frequently. $p_{loc}^{HQ}(base)$ increased, because popular contents have more priority.

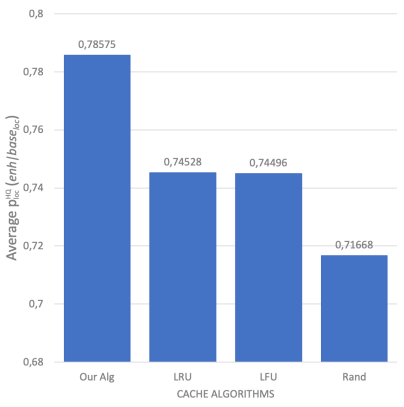

Fig. 8. The average $p_{loc}^{HQ}(enh|base_{loc})$ for algorithms

Our algorithm doesn't do anything special about the relationship between base and enhancement layer. The popularity of contents effect enhancement layers same as the base layer. Device cache storages have popular enhancement layers too. Furthermore, we replace old enhancement layers with new enhancement layers, likewise in bases. We don't mix different type replacements. In our algorithm, each content layer type struggle with contents with the same type. So enhancement layers are no invaded by base layers. In the other algorithm, the invasion of the base layers is very possible, because the mean of the base layer size is 5 times higher than the mean of the enhancement layer size.

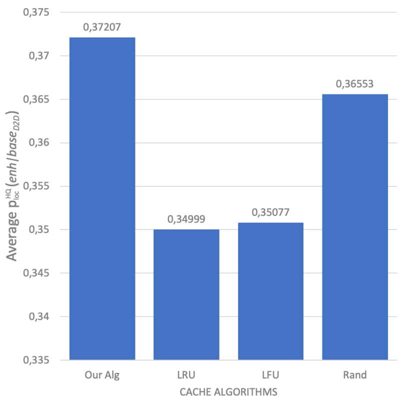

Fig. 9. The average $p_{loc}^{HQ}(enh|base_{D2D})$ for algorithms

We try to reduce access time to any content at each device in our algorithm. We create small local areas that contain each unique contents based on their popularity. Devices remove contents looking their local areas. So D2D service is okay for it. It thinks itself as part of the local area. It removes the content that most of its neighbor has this content. Finally our algorithm increase this rate.

V. CONCLUSIONS

Our algorithm distributes content caches homogeneously by their weights. The weight of the cache is calculated by dividing the count of it in the local area by its popularity. The higher weight means we have enough amount of the content in that area. It reduces distances, so it results in lower latency. It takes into account popularity, so local hit rates are improved. Improvements are smaller than we expected. We tried to change system configuration for better improvements such as N_c, T_{sim}, $cache_{dev}$, R_d. It might provide advantages at the different extreme situation. We couldn't be established better result improvements.