Ins and Outs: Optimal Caching and Re-Caching Policies in Mobile Networks

Wei Bao
School of Information Technologies, University of Sydney
wei.bao@sydney.edu.au

Dong Yuan
School of Electrical and Information Engineering, University of Sydney
dong.yuan@sydney.edu.au

Keqi Shi
School of Electrical and Information Engineering, University of Sydney
sherryskq@gmail.com

Weiyu Ju
School of Electrical and Information Engineering, University of Sydney
weju3435@uni.sydney.edu.au

Albert Y. Zomaya
School of Information Technologies, University of Sydney
albert.zomaya@sydney.edu.au

ABSTRACT

Caching is essential for data-intensive mobile applications to reduce duplicated data transmission. In this paper, we study the optimal probabilistic caching and re-caching policies in mobile networks as the file popularity may change over time. We propose a Probabilistic File Re-caching (PFR) policy to match the updated popularity. Through PFR, files cached (resp. not cached) are probabilistically opted out (resp. opted in). PFR is with substantial advantages: (1) PFR can handle a huge combinatorial number of all possible situations. (2) The expected number of replaced files is minimized. (3) The computational complexity of PFR is low. Second, we study a utility optimization problem in the mobile network, in order to optimally decide the probability that each file is cached and whether a file should be downloaded from a peer device or directly from the server. Even though the optimization problem is non-convex programming in nature, we devise a computationally efficient Optimal Probabilistic Caching and Requesting (OPCR) policy, through decoupling the decision variables, to derive a globally optimal solution. Finally, we develop a real-world prototype and conduct trace-driven simulations to validate and evaluate our proposed PFR and OPCR policies.

1 INTRODUCTION

Along with the development of mobile computing technologies, the ever increasing processing capacity of mobile devices enables more and more data-intensive applications, e.g., multimedia, augment reality (AR) applications, etc., to be available for mobile users. However, in the meantime, the increasing demand of downloading application data not only brings huge monetary cost to mobile users, but also causes significant delay due to the limited capacity of mobile networks that cannot handle the explosively growing traffic [34]. To cope with this challenge, smart caching strategies which enable *content reuse* have been investigated to reduce the duplicated data transmission. In particular, mobile devices themselves can send and receive data to/from each other in a peer-to-peer mode, without resorting to infrastructure support (e.g., WiFi or 4G) [28]. In this way, proactively caching popular contents in the mobile devices can substantially improve the network capacity and users' quality of experience.

In spite of the potential benefits, the design of an efficient caching policy, especially in the mobile environment faces additional challenges. A data-intensive application contains a huge amount of data in a content library, but the storage space of mobile devices could be quite limited. The total volume of contents in an AR game, for example, could be as large as hundreds of gigabytes, but the available storage for caching in a mobile device may be in the scale of a few gigabytes [11]. As a consequence, it is a must to determine how to wisely cache a portion of contents in each mobile device that can maximize the benefits of both the users' self-use and peer users' retrieving requests. Furthermore, the popularity of the contents may vary frequently. For example, AR game players' interests change from time to time and from location to location. An efficient updating policy is required to automatically adjust the cached contents to adapt to the popularity changes. Finally, even though mobile devices could directly request a content from its peers to avoid extra delay or cost of downloading from a remote server, such peer-to-peer mode should still be avoided if the communication distance is too long or the wireless channel state is poor. A smart mechanism to switch between the peer-to-peer and client-server modes is also required.

In this paper, we are motivated to address the aforementioned issues by proposing a set of efficient mobile caching and re-caching policies. (1) Referred to as the Probabilistic File Re-caching (PFR) policy, we address the file replacement problem as the file popularity changes. The core design is *probabilistic opt-in and opt-out*: each file in the cache (resp. not in the cache) is opted out (resp. opted in) with an easily computed probability, and the number of opt-out files equals to the number of opt-in files. PFR policy has to handle a combinatorial number of possible situations when an update occurs,[1] thus it is impractical to design PFR on a possibility-by-possibility basis. Through our proposed analysis, we prove that the huge number of possible situations can be characterized by a simple set of linear equations, which substantially reduces the complexity for subsequent design of PFR. Furthermore, whenever an update occurs, PFR realizes the update through only replacing the minimum number of cached files. (2) Referred to as the Optimal Probabilistic Caching and Requesting (OPCR) policy, we address how to optimally determine the probability that each file is cached

[1] In this paper, there are N files in total and K files can be cached, so that PFR must be able to handle $\binom{N}{K}$ possibilities since any K of the N files may be cached when an update occurs.

and whether a file should be downloaded from a peer device or directly from the server. The resultant optimization problem is of non-convex programming in nature and cannot be solved with a standard method. Instead, we explore an important structure in solving the problem, so that the decision variables can be decoupled and optimized one by one to derive a *globally optimal* solution.

Finally, we establish a real-world prototype, based on AllJoyn framework [1] and Android OS, in order to implement our proposed caching policies. Additional simulations are conducted based on the trace of the prototype. Both prototype experiments and simulations demonstrate PFR and OPCR can bring a substantial performance gain compared with conventional benchmark policies.

The rest of this paper is organized as follows: In Section 2, we present the design overview of caching and re-caching policies. In Section 3, we propose PFR policy and theoretically demonstrate its significant advantages. In Section 4, we propose OPCR policy through solving the optimization problem. In Section 5, we apply the prototype implementation and simulations to evaluate the proposed OPCR and PFR policies. Finally, prior related papers are discussed in Section 6 and the conclusions of this paper are given in Section 7.

2 DESIGN OVERVIEW OF MOBILE FILE CACHING SYSTEM

2.1 System Overview

We study caching policies in a mobile caching system. The system is comprised of multiple mobile peers. Each peer may request files in a library, e.g., a mobile game terminal requests specific game levels. Each peer itself may cache some files in its memory or disk. Whenever a peer requests a file that is not cached locally, the file can be downloaded from a neighboring peer or directly from the server with extra cost.

In this paper, given the nature of mobile networks, e.g., users' behaviour and movements are highly dynamic, we consider the independent probabilistic caching policy: Each file is cached with a caching probability, and each peer independently randomly caches files [3]. Peers make their own decisions without consulting each other and thus the policy is fully distributed. Please note that file caching with dependency may bring benefits. For example, if a file is cached at a peer, the peer's neighbor should avoid caching that file again to improve file diversity. However, such dependent caching causes additional coordination and communication among peers in the system, which is not suitable to systems considered in this paper with many relatively independent peers and potential frequent changes in file popularity. As a consequence, the independent probabilistic caching policy is the main focus of this paper.

The designed caching policy contains three sectors, as shown in Fig. 1. First, we discuss the Probabilistic File Caching (PFC) policy as a prior required mechanism, to determine how to cache the files at each peer, given that the file caching probabilities are calculated. However, PFC is insufficient to handle file replacement when the caching probabilities are updated (which is caused by changing popularity of files, to be discussed shortly). Therefore, we further design the Probabilistic File Re-caching (PFR) policy to efficiently update the cached file to match the updated probabilities. Finally, we propose the Optimal Probabilistic Caching and

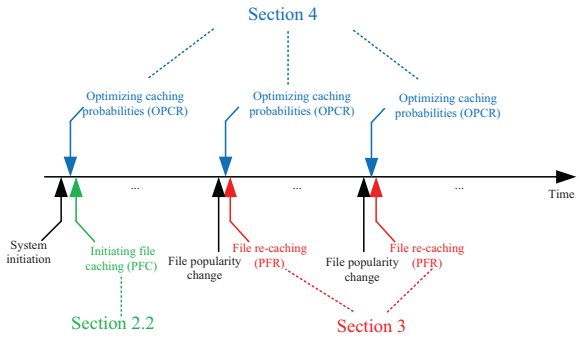


Figure 1: Summary of problems solved in this paper.

Requesting (OPCR) policy, to optimize the caching probabilities as well as to determine when to switch from peer-to-peer requesting to client-server requesting, based on the popularity of the files. In summary, when the *popularity* of files changes, it will further change the optimal *caching probabilities* computed by OPCR, and thus the files will be *re-cached* by PFR policy to achieve the updated (optimal) caching probabilities.

2.2 Probabilistic File Caching (PFC) Policy

In this section, we discuss the Probabilistic File Caching (PFC) policy, which determines which files are cached at each mobile peer, given the caching probability of each file. Throughout this paper, we assume that there are N files in the library in total. Let $\mathcal{F} = \{1, 2, \ldots, N\}$ denote the set of files. All files have the same size, normalized to 1. Please note that for the cases of unequal files sizes, each file can be split into chunks of equal size, so that the same analysis can still be applied [3]. Each mobile peer independently randomly caches exactly K files so that other peers may request cached files in a peer-to-peer fashion. K is a positive integer smaller than N. The jth file is cached with probability q_j. We define $\mathbf{q} \triangleq (q_1, q_2, \ldots, q_N)$, which is referred to as the caching probabilistic vector (CPV) in this paper. CPV is assumed to be fixed for this moment in this subsection.

Please note that PFC mechanism is proposed in [3], which is reviewed in this subsection for completeness, since all subsequent analyses are relevant to PFC. However, PFC does not address (1) how to update the files if CPV changes and (2) how to determine the CPV optimally. These two issues will be addressed in Sections 3 and 4 later, which are the two major contributions of this work.

PFC aims to realize the following two conditions via real-world implementation:

$$\sum_{j=1}^{N} q_j = K, \tag{1}$$

$$0 \le q_j \le 1, \forall j = 1, 2, \ldots, N. \tag{2}$$

Given (1)–(2), PFC realizes: (1) the jth file is cached with probability q_j; and (2) exactly K of the N files are cached. PFC is not a straightforward design as it achieves probabilistic caching in a *fixed-size* cache.

PFC is implemented as follows. First, we draw K horizontal virtual caching intervals (VCIs) from top to bottom. Each of the

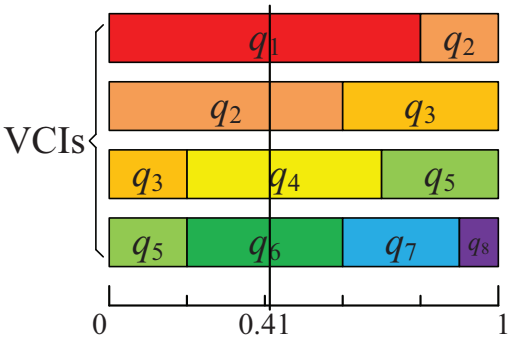


Figure 2: Policy with $N = 8$ and $K = 4$. $\mathbf{q} = (0.8, 0.8, 0.6, 0.5, 0.5, 0.4, 0.3, 0.1)$. $X = 0.41$. Files 1, 2, 4, and 6 are cached accordingly.

VCIs is with a unit length, ranging from 0 to 1 in x-coordinate, as shown in Fig. 2. Second, the VCIs will be occupied by the N files from left to right and from top to bottom. Files $1, 2, \ldots, N$ are visited subsequently. When the jth file is visited, a length of q_j of the next available VCI is occupied by this file, e.g., file 1 occupies $[0, 0.8]$ in the first VCI in Fig. 2. If this VCI does not have enough space, the file continues to fill the VCI underneath, e.g., file 2 occupies $[0.8, 1]$ in the first VCI and $[0, 0.6]$ in the second VCI in Fig. 2. The K VCIs are fully covered when all N files are visited since $\sum_{j=1}^{N} q_j = K$. Finally, we generate a random number X uniformly distributed in $[0, 1]$ and draw a vertical sweeping beam at the value X. The beam intersects with K VCIs. The K files intersected by the beam are selected as the cached files. Please note that the jth file occupies a length of q_j in one or two VCIs. X is randomly distributed in $[0, 1]$, so that the probability that X is within the range occupied by the jth file is q_j, and thus the caching probability of the jth file is q_j.

The computational complexity of the above PFC policy is $O(N)$. The complexity to determine the starting and ending points of each file's occupancy is $O(N)$. After generating the random sweeping beam, the complexity to determine if it intersects with each of the files is $O(N)$. Therefore, the overall complexity is $O(N)$.

3 PROBABILISTIC FILE RE-CACHING (PFR) POLICY

In this section, we consider the scenario where the CPV $\mathbf{q}$ is changed to $\mathbf{q}' = (q'_1, q'_2, \ldots, q'_N)$. This happens when the popularity changes, so that the CPV is also updated, as shown in Fig. 1.[2] Please note that the detailed derivation of CPV based on updated popularity will be discussed in Section 4. Then, the cached files should be replaced to match the new CPV. One straightforward way is to rerun the PFC policy, and all old cached files are replaced by the new selected files. However, this may lead to heavy file replacement even if the CPV is slightly changed. For example, in Fig. 2, suppose CPV is slightly changed to $(0.8001, 0.7999, 0.6, 0.5, 0.5, 0.4, 0.3, 0.1)$. If we rerun PFC policy and the random sweeping beam X equals to 0.95, then 2, 3, 5, and 8 are selected. Three out of four previously cached files 1, 4, and 6 are replaced.

[2]Please note that our model can accommodate the scenarios where new files enter the library: In vector $\mathbf{q}$, the entries corresponding to these new files are set to zero.

In reality, we may experience frequent slight changes in file popularity. If PFC policy is rerun each time, heavy file replacement could be triggered frequently. In order to address this issue, we are motivated to design an efficient re-caching policy so that the number of replaced files is minimized.

In this section, we design the PFR policy, to probabilistically re-cache files so that the new CPV can be achieved. For presentation convenience, let $\mathcal{F}_1$ denote the set of cached files prior to the re-cache event. $\mathcal{F}_1 = \{k_1, k_2, \ldots, k_K\}$ is a subset of $\mathcal{F}$. The cardinality of $\mathcal{F}_1$ is K. Let $\mathcal{F}_2 = \mathcal{F} \setminus \mathcal{F}_1 = \{l_1, l_2, \ldots, l_{N-K}\}$ denote the set of files not in the cache prior to the re-cache event. The cardinality of $\mathcal{F}_2$ is $N - K$.

Please note that as shown in Fig. 1. As time goes on, there are multiple re-cache events after the system initiation. The cached files are selected by PFC policy at the beginning, and PFR is used for all subsequent re-cache events to replace cached files.

3.1 PFR Design Objectives

In this subsection, we first list the design objectives of PFR policy as follows.

3.1.1 Adaptability. For *any two valid* CPVs $\mathbf{q}$ and $\mathbf{q}'$, PFR can realize the probability change from $\mathbf{q}$ to $\mathbf{q}'$.

3.1.2 Robustness. Since any K out of N files may be cached prior to a re-cache event, PFR can handle all $\binom{N}{K}$ situations.

3.1.3 Computational Efficiency. The computational complexity of PFR is low.

3.1.4 Minimum File Replacement. The expected number of replaced files in a re-cache event is minimized.

3.1.5 Insensitivity to Minor Variance. If the difference between the old and new CPVs is small, the cached files are changed with a small probability.

3.2 PFR Design Overview

In this subsection, we present the design of PFR policy to realize the aforementioned objectives. The first core design is called *probabilistic opt-in and opt-out*: Each file j is associated with an opt-out probability x_j and an opt-in probability y_j. When a re-cache event occurs, if the jth file is in the cache, it is removed from the cache with a probability of x_j; if it is not in the cache, it is cached with a probability of y_j (to replace removed ones). We define $\mathbf{x} \triangleq (x_1, x_2, \ldots, x_N)$ and $\mathbf{y} \triangleq (y_1, y_2, \ldots, y_N)$. Please note that these opt-out and opt-in probabilities $\mathbf{x}$ and $\mathbf{y}$ are designed values. They must satisfy several conditions to ensure successful file updates. The derivation of x_j and y_j will be discussed in Section 3.3 shortly.

The second core design is called *virtual interval opt-in and opt-out*, which is inspired by the PFC policy. However, different from PFC, the virtual intervals are not used to decide which files are cached, but to decide which files are opted in and out. The design is summarized as follows.

3.2.1 Opt-Out and Opt-In Intervals. We draw K horizontal *opt-out intervals* and K *opt-in intervals*. The opt-out intervals are used to determine which cached files are removed and the opt-in intervals

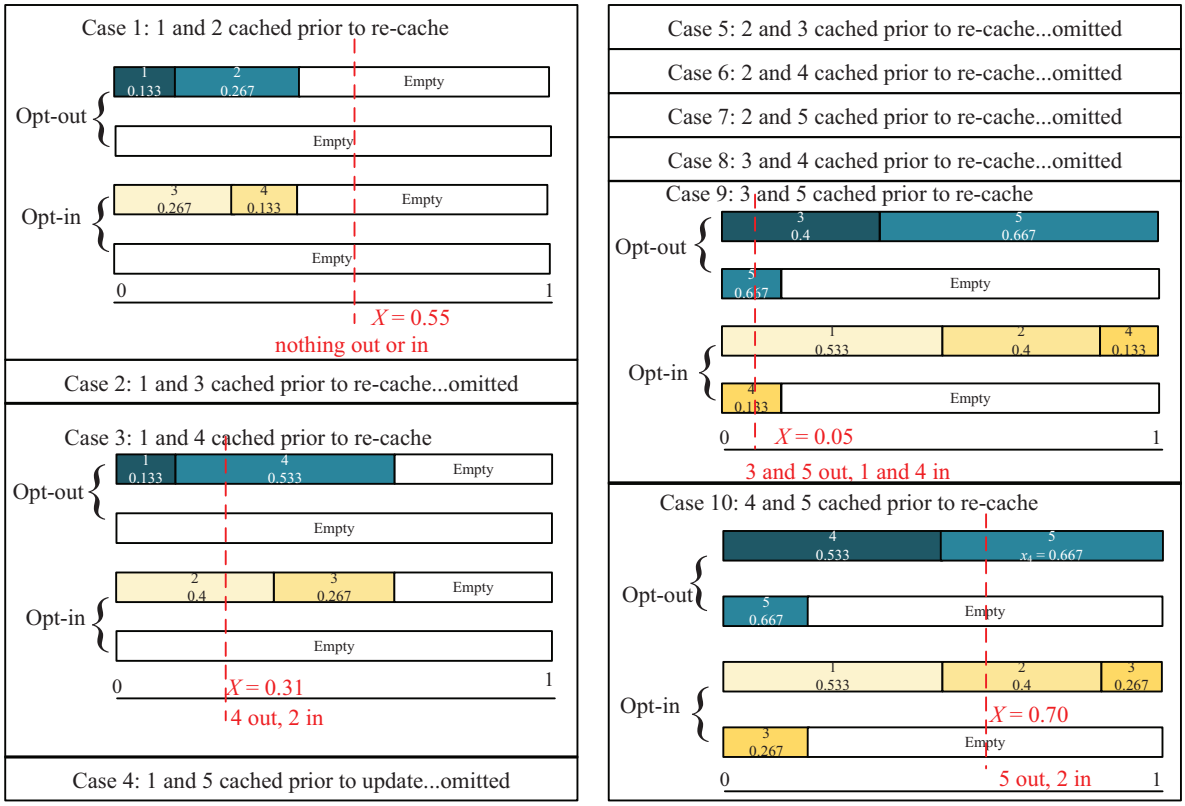


Figure 3: An example of PFR policy. $N = 5$ and $K = 2$. The old CPV is $(0.2, 0.3, 0.4, 0.5, 0.6)$ and the new CPV is $(0.6, 0.5, 0.4, 0.3, 0.2)$. The derived opt-out and opt-in probabilities are $(0.133, 0.267, 0.4, 0.533, 0.667)$ and $(0.533, 0.4, 0.267, 0.133, 0)$. There are $\binom{5}{2} = 10$ cases: files $(1, 2)$, $(1, 3)$, $(1, 4)$, $(1, 5)$, $(2, 3)$, $(2, 4)$, $(2, 5)$, $(3, 4)$, $(3, 5)$, or $(4, 5)$ are cached prior to the re-cache event. For briefness, we only show four cases for demonstration purpose. In Case 1, the random sweeping line $X = 0.55$, intersecting with the "empty" intervals, so that no files are opted out or opted in. In Case 3, X intersects with file 4 in the opt-out intervals and file 2 in the opt-in intervals so that file 4 is out and file 2 is in. Similarly, in Case 9, files 3 and 5 are out and files 1 and 4 are in. In Case 10, file 5 is out and 2 is in.

are used to determine which files are cached. Each of the intervals is with a unit length, ranging from 0 to 1 in x-coordinate, as shown in Case 1 in Fig. 3.

3.2.2 Interval Occupation. As shown in Case 1 in Fig. 3. The opt-out intervals will be occupied by the files already in the cache, i.e., $\mathcal{F}_1$, from left to right and from top to bottom. Files $k_1, k_2, \ldots, k_K$ are visited subsequently. When file k_j is visited, a length of x_{k_j} (i.e., its opt-out probability) of the next available opt-out interval is occupied by this file. If this opt-out interval does not have enough space, the file continues to fill the interval underneath. If all files are visited, the part of intervals unoccupied is marked *empty*. Similarly, the opt-in intervals will be occupied by the files not in the cache, i.e., $\mathcal{F}_2$. Still, the part of intervals unoccupied is marked empty.

3.2.3 Random Sweeping Beam and File Re-cache. We generate a random number X uniformly distributed in $[0, 1]$ and draw a vertical sweeping beam at the value X. The beam intersects with both opt-out intervals and opt-in intervals. Among the opt-out intervals, the files intersected by the beam are opted out. Among the opt-in intervals, the files intersected by the beam are opted in. Please note that the line may intersect with empty intervals, and these intersections will not cause opt-outs or opt-ins. PFR policy must guarantee that *the number of opt-out files equals to the number of opt-in files.* In order to achieve this, the total length of occupied (resp. empty) opt-out intervals must be equal to the total length of occupied (resp. empty) opt-in intervals. In other words, no matter where the sweeping beam is, it always intersects with the same number of opt-out files and opt-in files. This can be achieved by careful design of opt-out and opt-in probabilities, to be discussed in Section 3.3. As shown in Case 1 in Fig. 3, the total length of occupied opt-out and opt-in intervals is 0.4. If X is in $[0, 0.4)$, then one file is opted out and one file is opted in. If X is in $[0.4, 1)$, no file is opted out or in.

The computational complexity to implement the above PFR policy is $O(N)$. The complexity to determine the starting and ending points of each file's occupancy in opt-in and opt-out intervals is $O(N)$. After generating the random sweeping beam, the complexity to determine if it intersects with each file is $O(N)$. Therefore, the overall complexity is $O(N)$.

3.3 Design of Opt-Out and Opt-In Probabilities

As discussed in Section 3.2, the opt-out and opt-in probabilities $\mathbf{x}$ and $\mathbf{y}$ are important parameters to realize PFR policy. In this subsection, we derive these probabilities. We first list the conditions that $\mathbf{x}$ and $\mathbf{y}$ must satisfy. Then we derive closed-form expressions of $\mathbf{x}$ and $\mathbf{y}$. Meanwhile, we show several important properties of $\mathbf{x}$ and $\mathbf{y}$, so that PFR policy achieves the objectives in Section 3.1. For presentation convenience, let $z_i = 1 - x_i$ denote the stay-in probability, i.e., the probability that a file stays in the cache when a re-cache event occurs. Let $\mathbf{z} \triangleq (z_1, z_2, \ldots, z_N)$. There is a one-to-one mapping between $\mathbf{x}$ and $\mathbf{z}$ so that we focus on the derivation of $\mathbf{z}$ instead of $\mathbf{x}$.

3.3.1 Conditions on $\mathbf{z}$ *and* $\mathbf{y}$. First, in order to guarantee the caching probabilities are changed from $\mathbf{q}$ to $\mathbf{q}'$, the following law of total probability must be satisfied

$$q_i z_i + (1 - q_i) y_i = q_i', \forall i. \tag{3}$$

Please note (3) represents a set of N equations.

Second, as discussed in Section 3.2, when a re-cache event occurs, the number of opt-out files equals to the number of opt-in files in any situation. Therefore, for any K files cached prior to the re-cache event, the sum of opt-out probabilities of these K files equals to the sum of opt-in probabilities of the rest $N-K$ files. As a consequence, we have

$$\sum_{i \in \mathcal{G}} (1 - z_i) = \sum_{i \in \mathcal{F} \setminus \mathcal{G}} y_i, \forall \mathcal{G} \subset \mathcal{F} \text{ and } |\mathcal{G}| = K. \tag{4}$$

where $\mathcal{G}$ denotes an arbitrary subset of $\mathcal{F}$ with K elements.

3.3.2 Robustness of PFR. (4) represents a set of $\binom{N}{K}$ equations. This is because the set of cached files prior to a re-cache event can be any K files, and the PFR policy must be able to handle all of these $\binom{N}{K}$ situations.

Please note that if the re-cache event is the first one, the cached files prior to the re-cache event are selected by the PFC policy. In the example in Fig. 2, files 7 and 8 cannot be cached together by the PFC policy since the random sweeping beam cannot intersect with files 7 and 8 simultaneously. However, we need to emphasize that the PFR policy is not only used for the first re-cache event. It must be able to handle all subsequent re-cache events – any K out of N files may be cached prior to a re-cache event. In Fig. 2, it is possible that file 7 is cached at the beginning and file 8 is opted in later, so that files 7 and 8 may still be cached together prior to a re-cache event later. In summary, we conclude that (4) ensures the robustness of PFR policy.

3.3.3 Derivation of $\mathbf{z}$ *and* $\mathbf{y}$. In order to compute the values of $\mathbf{z}$ and $\mathbf{y}$, solving equations (3) and (4) directly is not practical. There are two major issues: (1) solving $\binom{N}{K} + N$ equations involves very high computational complexity; and (2) the number of equations is much greater than the number of unknown variables so that solutions may not exist. In what follows, by further examining (4), we show that these $\binom{N}{K}$ equations are equivalent to N equations shown in the following theorem:

Theorem 3.1. *The set of equations (4) is equivalent to the following N equations*

$$\sum_{i=1}^{K} (1 - z_i) = \sum_{i=K+1}^{N} y_i, \tag{5a}$$

$$z_j - y_j = z_{j+1} - y_{j+1}, \forall j = 1, 2, \ldots, N-1. \tag{5b}$$

Proof. **Part (a), (4)⇒(5)**

From (4), let $\mathcal{G} = \{1, 2, \ldots, K\}$. We have $\sum_{i=1}^{K}(1-z_i) = \sum_{i=K+1}^{N} y_i$ so that (5a) is true.

From (4), let $\mathcal{G}$ be an arbitrary K-element set such that $j \in \mathcal{G}$ and $j+1 \notin \mathcal{G}$. Let $\mathcal{G}' = \mathcal{G} \setminus \{j\} \bigcup \{j+1\}$. Therefore, we have

$$\sum_{i \in \mathcal{G}} (1 - z_i) = \sum_{i \in \mathcal{F} \setminus \mathcal{G}} y_i, \tag{6}$$

$$\sum_{i \in \mathcal{G}'} (1 - z_i) = \sum_{i \in \mathcal{F} \setminus \mathcal{G}'} y_i. \tag{7}$$

Subtracting (6) from (7) in both left hand side (LHS) and right hand side (RHS) leads to $z_j - y_j = z_{j+1} - y_{j+1}$, i.e., (5b).

Part (b), (5)⇒(4)

(5b) implies that

$$z_j - y_j = z_k - y_k, \forall j, k. \tag{8}$$

Equivalently, we have

$$z_j - z_k = y_j - y_k, \forall j, k. \tag{9}$$

Let $\mathcal{G}$ be an arbitrary subset of $\mathcal{F}$ with K elements. Let $\mathcal{G}_0 = \{1, 2, \ldots, K\} \setminus \mathcal{G}$ denote the set of elements in $\{1, 2, \ldots, K\}$ but not in $\mathcal{G}$. Let $\mathcal{G}_1 = \mathcal{G} \setminus \{1, 2, \ldots, K\}$ denote the set elements in $\mathcal{G}$ but not in $\{1, 2, \ldots, K\}$. Please note that $|\mathcal{G}_0| = |\mathcal{G}_1|$ because both $\mathcal{G}$ and $\{1, 2, \ldots, K\}$ have K elements.

From (5a), we have

$$\sum_{i=1}^{K} (1 - z_i) = \sum_{i=K+1}^{N} y_i. \tag{10}$$

Then,

$$\sum_{i=1}^{K} (1 - z_i) + \sum_{j \in \mathcal{G}_0} z_j - \sum_{j \in \mathcal{G}_1} z_j = \sum_{i=K+1}^{N} y_i + \sum_{j \in \mathcal{G}_0} z_j - \sum_{j \in \mathcal{G}_1} z_j. \tag{11}$$

The LHS of (11) becomes

$$\sum_{i=1}^{K} (1 - z_i) + \sum_{j \in \mathcal{G}_0} z_j - \sum_{j \in \mathcal{G}_1} z_j = \sum_{i \in \mathcal{G}} (1 - z_i). \tag{12}$$

From (9), since $|\mathcal{G}_0| = |\mathcal{G}_1|$, we have

$$\sum_{j \in \mathcal{G}_0} z_j - \sum_{j \in \mathcal{G}_1} z_j = \sum_{j \in \mathcal{G}_0} y_j - \sum_{j \in \mathcal{G}_1} y_j. \tag{13}$$

The RHS of (11) becomes

$$\begin{aligned} &\sum_{i=K+1}^{N} y_i + \sum_{j\in\mathcal{G}_0} z_j - \sum_{j\in\mathcal{G}_1} z_j \qquad (14)\\ =&\sum_{i=K+1}^{N} y_i + \sum_{j\in\mathcal{G}_0} y_j - \sum_{j\in\mathcal{G}_1} y_j\\ =&\sum_{i\in\mathcal{F}\setminus\mathcal{G}} y_i. \end{aligned}$$

Therefore, by combining (11), (12), and (14), we have

$$\sum_{i\in\mathcal{G}}(1-z_i) = \sum_{i\in\mathcal{F}\setminus\mathcal{G}} y_i. \qquad (15)$$

□

Please note Theorem 3.1 holds because many of the equations in (4) are linearly dependent. After reducing the linear dependency, only N equations are left. Therefore, Theorem 3.1 is a key step to ensure the tractability and computational efficiency in solving $\mathbf{z}$ and $\mathbf{y}$ values.

In the next step, we aim to derive $\mathbf{z}$ and $\mathbf{y}$ by solving equations (3) and (5). To achieve so, one straightforward way is to employ Gaussian elimination. However, this method is still limited for the following two reasons: (1) The computational complexity of Gaussian elimination, i.e., $O(N^3)$, is still not low enough. (2) The solutions are not in closed form so that it is not easy to further characterize the solutions, e.g., to prove that the solution is in the range of $[0, 1]$. Therefore, we further explore expressions (3) and (5) and successfully derive a closed-form solution in the following theorem:

Theorem 3.2. *Let C be an arbitrary (real) number. (16) is a solution to (3) and (5).*

$$\begin{cases} z_i = C(1-q_i) + q'_i, \forall i,\\ y_i = q'_i - Cq_i, \forall i. \end{cases} \qquad (16)$$

Proof. It is straightforward to verify through substituting (16) into (3) and (5). □

In (16), we observe that the solution to (3) and (5) is not unique, i.e., C could be an arbitrary number at the current stage. This is because there is still linear dependency in the linear equations (3) and (5). Among the solutions, we further need to find those ones satisfying $0 \le z_i \le 1, 0 \le y_i \le 1, \forall i$ since z_i and y_i values are probabilities. By letting $C = 0$, we have $z_i = q'_i, y_i = q'_i, \forall i$, which is always a valid solution since q'_i values are all in the range of $[0, 1]$. Therefore, we reach the following proposition:

Proposition 3.3 (Adaptability). *For any valid old and new CPVs $\mathbf{q}$ and $\mathbf{q}'$, we can always find opt-out and opt-in probabilities satisfying (3) and (4). Therefore, the PFR policy is adaptive.*

3.3.4 Min File Replacement and Insensitivity to Minor Changes. Since there are multiple solutions to (3) and (5), we are motivated to further derive the best one. Recall that one of our aims is to minimize the opt-out and opt-in probabilities. Also, when the new CPV is very close to the old CPV, the files should be opted out and in with small probabilities. In order to achieve these goals, we can adjust the C value to minimize the opt-out and opt-in probabilities. From (16), we notice that if C is increased, all opt-out and opt-in probabilities will decrease. However, all of these probabilities should be in the range of $[0, 1]$. Therefore, we set

$$C = \min_i \left[\min\left(\frac{1-q'_i}{1-q_i}, \frac{q'_i}{q_i}\right)\right], \qquad (17)$$

which is the maximum C that satisfies the condition that z_i and y_i are in the range of $[0, 1]$. Therefore, we have

Proposition 3.4 (Minimum File Replacement). *Substituting (17) into (16), all opt-out and opt-in probabilities are minimized. Therefore, the expected number of replaced files in a re-cache event is minimized.*

Through substituting (17) into (16), we also observe that when $\mathbf{q} \to \mathbf{q}'$, we have $C \to 1$ and thus $\mathbf{z} \to \mathbf{1}$[3] and $\mathbf{y} \to \mathbf{0}$. This means that if the new CPV approaches to the old CPV, the opt-out and opt-in probabilities approach to zeros.

We define *CPV changing index* η as the minimum positive value satisfying $1-\eta \le \frac{q'_i}{q_i} \le 1+\eta$ and $1-\eta \le \frac{1-q'_i}{1-q_i} \le 1+\eta, \forall i$. η is used to measure the difference between the old and new CPVs. $\eta = 0$ implies that the old and new CPVs are the same. Please note that $1-\eta \le C$ by definition in (17). Then, we have

$$y_i = q'_i - Cq_i \le (1+\eta)q_i - (1-\eta)q_i = 2\eta q_i. \qquad (18)$$

Also, we have

$$z_i = C(1-q_i) + q'_i \ge (1-\eta)(1-q_i) + (1-\eta)q_i = 1-\eta. \qquad (19)$$

As a consequence, we can see that the stay-in and opt-in probabilities are bounded by η. Smaller η leads to smaller y_i and larger z_i. $y_i \to 0$ and $z_i \to 1$ if we have $\eta \to 0$. Therefore, we reach the following proposition:

Proposition 3.5 (Insensitivity to Minor Variance). *The stay-in and opt-in probabilities are bounded by (18) and (19). If the new CPV approaches to the old one, the opt-out and opt-in probabilities approach to zeros.*

3.3.5 Analysis on Computational Complexity. Based on the above analysis, we notice that the computational complexity to derive C in (17) is $O(N)$ and the computational complexity to derive $\mathbf{z}$ and $\mathbf{y}$ in (16) is also $O(N)$, so that the derivation of the opt-out and opt-in probabilities are computationally efficient. Please note that our proposed approach to compute $\mathbf{z}$ and $\mathbf{y}$ substantially reduces the computational complexity. Through the linear dependency reduction, the number of equations in the original problem is reduced from $\binom{N}{K} + N$ to $2N$. We further find the closed-form solution (16)–(17), so that numerically solving linear equations, including Gaussian elimination with a computational complexity of $O(N^3)$, can be avoided.

Please note that given $\mathbf{z}$ and $\mathbf{x}$, the computational complexity to implement PFR is $O(N)$, as discussed in Section 3.2. As a conclusion, we have the following proposition:

Proposition 3.6 (Computational Complexity). *The overall computational complexity of PFR is $O(N)$.*

[3]Here, $\mathbf{1}$ indicates a vector with all-1 entries.

3.4 Immediate and Delayed File Re-Caching

After deriving the opt-out and opt-in probabilities and conducting the PFR policy, the opt-out and opt-in files are determined. Then we have two choices to realize the file re-caching, immediate file re-caching (IFR) and delayed file re-caching (DFR) in reality. For IFR, the device downloads and caches the opt-in files and deletes the opt-out files immediately. For DFR, the device downloads and caches the opt-in files only if the device itself needs that file. An opt-out file is deleted when an opt-in file comes. If there are multiple opt-out files, we may employ approaches such as least frequently used (LFU) eviction, least recently used (LRU) eviction, etc. to determine which opt-out file is deleted first. Other mobile peers may still be able to request an opt-out file from this device as long as that file has not been deleted.

4 OPTIMAL PROBABILISTIC CACHING AND REQUESTING (OPCR) POLICY

4.1 Optimization Problem Formulation

In this section, we focus on the optimization of CPV $\mathbf{q}$, based on the file popularity. Please note that since the popularity may change, the resultant optimal CPV will be updated accordingly. Then PFR policy, proposed in Section 3 will be adopted to realize the updated CPV. Different from the scope of Section 3, we study how to derive $\mathbf{q}$ based on given file popularity in this section. In this paper, we assume that the popularity of the files is given as a priori. For example, the popularity can be estimated by counting the number of recent file requests to fit to the Zipf distribution [6, 14]. More complicated and accurate popularity estimation is out of the scope of this work.

We assume that the jth file is requested with a probability of p_j (i.e., independent reference model [12, 22]). p_j indicates the popularity of the file, i.e., larger p_j implies higher popularity. We have $\sum_{j=1}^{N} p_j = 1$. Let $\mathbf{p} \triangleq (p_1, p_2, \ldots, p_N)$ denote the popularity vector. In the analysis, we consider a general scenario where the request probabilities could be arbitrary.

Once a peer requests a file, it will first check its own cache; if the file is not cached, it will request to its nearest neighbor; if the nearest neighbor does not cache the requested file either, the 2nd nearest neighbor is requested; if still not found, the 3rd, 4th, ... nearest neighbors are requested; this procedure continues until (1) the file is found, or (2) the Lth nearest neighbor is requested but the file is still not found. Under (2), the peer initiates a connection to the Internet (e.g., via a cellular network), and requests the file directly from the server. The value L is referred to as *request limit*, which is to be optimized in this section. The file retrieved from neighbors or the remote server will be discarded when the user completed using the file.

Let R_l, $l = 1, 2, \ldots$, be the non-negative average utility of the user if a requested file is found at the lth closet peer. Let R_0 be the utility that the requested file is found at itself; and S be the utility of downloading the file directly from the server (when the file is not found at any nearby peer). Here the term utility denotes a "general utility" encompassing factors such as data rate, delay, power consumption, monetary cost, etc.[4] We do not specify the utility to keep our model more general. In our evaluation in Section 5, the utility is specified as delay performance.

We assume that $R_0 > R_1 > R_2 > \ldots$ since downloading a file from a closer peer is more beneficial than downloading it from a farther one. Also, we assume $R_l \to 0$ if l is large. Downloading a file from a very far peer (e.g., out of communications range) does not give any benefit. In additional, we have $R_0 > S > 0$, i.e., directly finding a file at its own cache is better than downloading from the server.

We aim to jointly optimize the caching probabilities $\mathbf{q}$ and the request limit L, to maximize the average user utility. If the user requests the jth file, the average utility is $\sum_{l=0}^{L} R_l(1-q_j)^l q_j + S(1-q_j)^{L+1}$. Thus, the overall average user utility is $\sum_{j=1}^{N} p_j \left(\sum_{l=0}^{L} R_l(1-q_j)^l q_j + S(1-q_j)^{L+1} \right)$ due to the law of total probability. As a result, we reach the following optimization Problem **P**

$$\max_{\mathbf{q}, L} \sum_{j=1}^{N} p_j \left(\sum_{l=0}^{L} R_l(1-q_j)^l q_j + S(1-q_j)^{L+1} \right), \tag{20}$$

$$\text{subject to} \sum_{j=1}^{N} q_j = K, \tag{21}$$

$$0 \le q_j \le 1, \forall j = 1, 2, \ldots, N, \tag{22}$$

$$L \text{ is a non-negative integer}, \tag{23}$$

where (21) and (22) are from (1) and (2). For presentation convenience, let $\text{obj}(\mathbf{q}, L)$ denote the objective function in (20).

The solution to Problem **P** is not straightforward. The challenge is two-fold. First, L is an integer, so that Problem **P** involves integer programming. Second, even if L is given, the remaining problem to optimize $\mathbf{q}$ is still non-convex. For an integer programming and non-convex optimization problem, it is not guaranteed that there is an algorithm that can find a globally optimal solution within a polynomial computational complexity. In what follows, we manage to develop a new approach through decoupling L and $\mathbf{q}$, so that a globally optimal solution to Problem **P** can be derived efficiently.

4.2 Derivation of Optimal L

In this subsection, we manage to decouple L and $\mathbf{p}$, so that L can be optimized first. We derive the following theorem:

Theorem 4.1. *Let L^* be the L value satisfying the following inequality: $R_{L+1} \le S < R_L$. Then, L^* is an optimal solution to Problem **P**.*

Proof. First, since $R_0 > R_1 > R_2 > \ldots$, $\lim_{L\to\infty} R_L = 0$, and $R_0 > S > 0$, there exists a unique L satisfying $R_{L+1} \le S < R_L$.

Second, if $L \ge L^*$, we have

$$\text{obj}(\mathbf{q}, L+1) - \text{obj}(\mathbf{q}, L) \tag{24}$$

$$= \sum_{j=1}^{N} p_j (R_{L+1} - S)(1-q_j)^{L+1} q_j \le 0. \tag{25}$$

[4] This model can also maximize "hit rate". The utility of a "hit" (local download) is 1 and the utility of a "miss" (remote download) is 0.

where (25) is because $R_{L+1} - S \leq 0$. Therefore, obj($\mathbf{q}, L$) is a non-increasing function of L if $L \geq L^*$.

Third, if $L < L^*$, we have

$$\text{obj}(\mathbf{q}, L+1) - \text{obj}(\mathbf{q}, L) \tag{26}$$

$$= \sum_{j=1}^{N} p_j (R_{L+1} - S)(1-q_j)^{L+1} q_j \geq 0. \tag{27}$$

where (27) is because $R_{L+1} - S > 0$. Therefore, obj($\mathbf{q}, L$) is a non-decreasing function of L if $L \leq L^*$.

As a consequence, for an arbitrary $\mathbf{q}$, obj($\mathbf{q}, L$) is a non-decreasing function of L if $L \leq L^*$, and a non-increasing function of L if $L \geq L^*$, and thus L^* is an optimal solution to Problem **P**. □

Theorem 4.1 shows that under any $\mathbf{q}$ value, L^* is always an optimal solution. As a consequence, we can derive L^* first without considering $\mathbf{q}$. Then, the optimal $\mathbf{q}$ is derived based on the given L^*, in the next subsection.

4.3 Derivation of Optimal q

After deriving the optimal L, we next derive the optimal $\mathbf{q}$ given $L = L^*$. As a consequence, we need to solve the optimization Problem **P'** as follows

$$\max_{\mathbf{q}} \sum_{j=1}^{N} p_j \left(\sum_{l=0}^{L^*} R_l (1-q_j)^l q_j + S(1-q_j)^{L^*+1} \right), \tag{28}$$

$$\text{subject to} \sum_{j=1}^{N} q_j = K, \tag{29}$$

$$0 \leq q_j \leq 1, \forall j = 1, 2, \ldots, N. \tag{30}$$

Theorem 4.2. *The objective function in (28) is concave in* $\mathbf{q}$.

Proof. First, let

$$F_j(q_j) \triangleq \sum_{l=0}^{L^*} R_l (1-q_j)^l q_j + S(1-q_j)^{L^*+1} \tag{31}$$

$$= \sum_{l=0}^{L^*} \left[-R_l (1-q_j)^{l+1} + R_l (1-q_j)^l \right] + S(1-q_j)^{L^*+1}$$

$$= R_0 + \sum_{l=0}^{L^*-1} (R_{l+1} - R_l)(1-q_j)^{l+1} + (S - R_{L^*})(1-q_j)^{L^*+1}.$$

Then, its first-order derivative is

$$\frac{\partial F_j(q_j)}{\partial q_j} = \sum_{l=0}^{L^*-1} (l+1)(R_l - R_{l+1})(1-q_j)^l + (R_{L^*} - S)(L^*+1)(1-q_j)^{L^*}. \tag{32}$$

Since $R_l > R_{l+1}$ and $R_{L^*} > S$, $\frac{\partial F_j(q_j)}{\partial q_j}$ is a decreasing function of q_j so that $\frac{\partial^2 F_j(q_j)}{\partial q_j^2}$ is non-positive. Because obj($\mathbf{q}, L^*$) $= \sum_{j=1}^{N} p_j F_j(q_j)$, obj($\mathbf{q}, L^*$) is a concave in $\mathbf{q}$.

□

Through Theorem 4.2, we show that Problem **P'** is a standard convex optimization problem. Therefore, it can be solved computationally efficiently by a standard convex optimization method, such as interior point method [5, Chapter 11]. Let $\mathbf{q}^*$ denote the optimal solution to Problem **P'**. Then $(\mathbf{q}^*, L^*)$ is an optimal solution to the original Problem **P**.

Remark 1. *Problem* ***P'*** *is convex when L is fixed at* L^**. If L is not fixed,* obj($\mathbf{q}, L$) *may not be concave in* $\mathbf{q}$*. Without decoupling L and* $\mathbf{q}$ *in Theorem 4.1, the original Problem* ***P*** *is non-convex and cannot be solved by a standard method.*

In summary, in this section, we derive the optimal CPV based on the file popularity. Even though the optimization problem is non-convex, we devise a computationally efficient OPCR policy to derive a globally optimal solution. By combining the OPCR and PFR policies, the system is adaptive to changes in file popularity. OPCR is implemented to derive the new optimal CPV, and then PFR is implemented to realize the updated CPV.

5 PERFORMANCE EVALUATION

5.1 Prototype

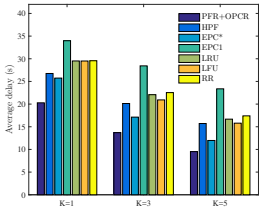
We develop a prototype mobile game system that realizes the proposed caching and re-caching policies in Android smartphones. In the prototype, i.e., the Android game application, we define 11 game levels ($N = 11$) which require different data files to be loaded before playing. The file size of each game level is 5 MB. Users can freely choose any level to play. If not cached locally, the corresponding data file will be downloaded either from nearby players or from the remote server. We adapt an open source software framework, AllJoyn [1], to realize the file transmission among users. In the AllJoyn framework, mobiles can discover each other to form a peer-to-peer network to distribute game contents.

To conduct experiments, we place nine different Android devices in an outdoor environment, including three Huawei P8 Lite phones, three Huawei MediaPad T1 tablets, one Huawei MediaPad M2 tablet, one Samsung Galaxy J5 phone, and one Meizu MX5 phone. The diversity of devices emulates users in a real mobile game environment. We setup a WiFi access point with the D-link DSL-2750U router. We add the nine devices to the WiFi network and place them in a line with equal 10-meter intervals. The performance of the Huawei P8 Lite phone placed at the end point of the line is investigated.

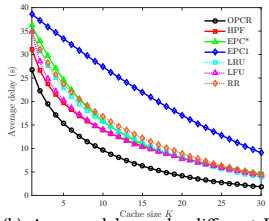
5.2 Evaluation Setup

5.2.1 Definition of Utility . The user utility is inversely related to the file downloading delay. We set $R_l = D_{\max} - D_l$, where D_l denotes the average file downloading delay from the lth closest peer, and $D_{\max}$ is the max possible downloading delay when a file is downloaded from the farthest peer (8th closest peer in the experiment). D_s is the file downloading delay from the server and $S = D_{\max} - D_s$. Through this setting, maximizing the average utility is equivalent to minimizing the average downloading delay. Table 1 shows the average delays in our experiment.

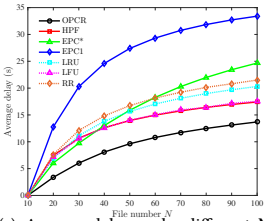
5.2.2 File Popularity. The file popularity is set to follow the Zipf distribution [6]. The kth most popular file is with a request probability of $p_k = \frac{k^{-\gamma}}{\sum_{j=1}^{N} j^{-\gamma}}$. We set $\gamma = 1$. File popularity updates frequently. The popularity of two (randomly selected) files are swapped every 10 requests.



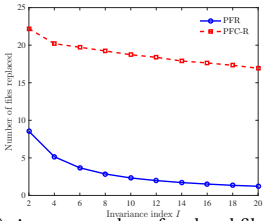
(a) Average delay performance by experiment.



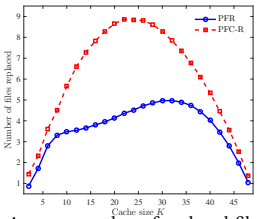
(b) Average delay under different K.



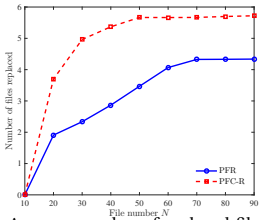
(c) Average delay under different N.



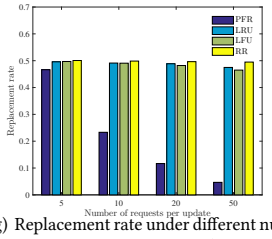
(d) Average number of replaced files in one popularity update event under different invariance indices.



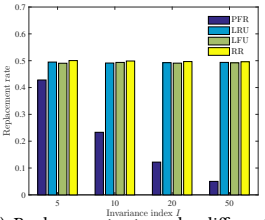
(e) Average number of replaced files in one popularity update event under different K.



(f) Average number of replaced files in one popularity update event under different N.



(g) Replacement rate under different numbers of requests per update (update frequency).



(h) Replacement rate under different invariance indices (update intensity).

Figure 4: Evaluation results.

Table 1: Summary of Delays (in second).

D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8 ($D_{\max}$)	D_s
11.2	11.3	11.6	12.6	15.9	18.8	50.1	75.0	40.0

5.2.3 Benchmark Policies. We compare our proposed PFR+OPCR policy with the following benchmark policies.

Highest Population First (HPF): Each peer caches K files with the highest popularity. Please note that through this policy, peers will not help each other since every peer caches the same K files.

Equal Probabilistic Caching (EPC): Each file is cached with the same probability $\frac{K}{N}$. In addition, EPC* denotes the sub-policy that the request limit L is optimized through our approach in Section 4.2. EPC1 denotes the sub-policy that the request limit is equal to 1.

We also compare with the other conventional non-probabilistic caching policies. Each peer individually implements Least Recently Used (LRU), Least Frequently Used (LFU), or Random Replacement (RR) policy.

5.3 Experiment Results

Fig. 4(a) shows the experiment results of the average file downloading delays run by the aforementioned AllJoyn prototype. The delay performance of PFR+OPCR and benchmark policies are presented under $K = 1$, $K = 3$, and $K = 5$. In this experiment, we observe that PFR+OPCR gives the best performance, followed by EPC*, HPF, LFU, LRU, RR, and EPC1. Even compared with the second best policy EPC*, PFR+OPCR reduces the delay by more than 25%. Please note that EPC* is partially based on our optimization in Section 4.2. The performance of other policies is even worse. In summary, our proposed PFR+OPCR brings substantial performance gain compared with non-optimal benchmark policies for the prototype.

5.4 Trace-Driven Simulation Results

In what follows, we adopt trace-driven simulation approach to demonstrate the benefits of our proposed OPCR and PFR policies under a wider range of system settings. The downloading delays from the lth closest peer and from the server are based on the experiment results in Table 1.

5.4.1 Performance of OPCR Policy. In Figs. 4(b)–4(c), we show the delay performance of OPCR and benchmark policies under different K and N values. File popularity follows Zipf distribution with $\gamma = 1$, and it does not change over time. In Fig. 4(b), we set $N = 50$. In Fig. 4(c), we set $K = 10$. From the figures, we shows that OPCR substantially outperforms all benchmark policies. We also observe that OPCR outperforms EPC*, and EPC* outperforms ECP1, illustrating that both the optimized L (Section 4.2) and optimized $\mathbf{q}$ (Section 4.3) effectively improve the delay performance. In addition, the performance of non-probabilistic caching policies (LRU, LFU, and RR) is close to that of EPC* and HPC, but much worse than that of OPCR. This result suggests that optimizing caching probabilities via OPCR is the key factor that escalates the performance of probabilistic caching policy to beat conventional non-probabilistic caching policies.

5.4.2 Performance of PFR Policy. We furthermore discuss the performance of PFR policy proposed in Section 3. In Fig. 4(d), we investigate the expected number of replaced files when CPV changes. We set $K = 50$ and $N = 100$. For comparison, we also show the performance of PFC-R: In each re-cache event, PFC is rerun and all old cached files are replaced by the new selected files. In the simulation, we randomly generate the old CPVs and new CPVs, and the largest caching probability difference cannot be greater than $\frac{1}{I}$. I is referred to as the *invariance index*. Larger I indicates smaller difference between the old and new CPVs. In Fig. 4(d), we observe that the average number of replaced files by PFR is

much smaller than that through PFC-R. When I keeps increasing, the average number of replaced files by PFR approaches to zero, matching our expectation. However, the PFC-R keeps replacing more than one third of the cached files. Especially, when $I = 20$ (i.e., a caching probability is changed by 0.05 at most), only around 1 file is replaced by PFR, but around 17 files (one third of the cached files) are replaced by PFC-R, showing that PFR can substantially reduce the workload of file replacement. In Figs. 4(e)–4(f), we study how many files are replaced if the file popularity changes. In each re-cache event, the popularity of two (randomly selected) files are swapped; then OPCR is rerun to compute the new optimal CPV; and finally PFR or PFC-R is implemented to re-cache files. We set $N = 50$ in Fig. 4(e) and $K = 10$ in 4(f). Both of figures show that PFR replaces much fewer files compared with PFC-R, further validating the usefulness and benefits of our proposed PFR policy.[5]

If Figs. 4(g)–4(h), we show the file replacement performance of PFR and conventional non-probabilistic caching policies (LRU, LFU, and RR). Please note that we have already shown that PFR+OPCR outperforms LRU, LFU, and RR in terms of delay performance. We further aim to show that PFR also results in fewer number of replaced files. We define the *replacement rate* as the average number of replaced files per request. The term "number of requests per update" indicates the number of file requests between two popularity update events (showing the frequency of popularity updates). In Fig. 4(g), the invariance index is fixed at 10. In Fig. 4(h), the number of requests per update is fixed at 10. From the figures, we clearly observe that the replacement rate of PFR decreases substantially as number of requests per update increases (update frequency decreases) or invariance index increases (update intensity decreases). This is because PFR replaces files only when popularity is updated, and the number of replaced files is small if the popularity is only slightly changed. Different from PFR, non-probabilistic caching policies replace files whenever there is a "miss" no matter if the popularity changes. Therefore, we notice that non-probabilistic caching policies are almost insensitive to frequency and intensity of popularity updates. Their replacement rate is high even if the popularity changes slightly or infrequently. In sum, the results demonstrate that PFR leads to much lower replacement rate compared with LRU, LFU, and RR.

6 RELATED WORK

Caching is a well recognized approach to improve performance of networks and distributed systems. Extensive research about caching policies has been conducted. However, due to the dynamic nature of mobile networks, re-caching policies have to be applied to deal with file updates, which have not been well investigated in academia.

For a single caching node, the authors of [20] propose a dynamic cache replacement scheme through learning the popularity of files in an online fashion. The paper [29] studies load balancing and caching performance in a hash partitioning cache. The transient and steady-state behavior of a family of list-based cache replacement algorithms is characterized in [13]. For a set of connected caching servers, the performance of several typical caching policies (e.g., LFU, LRU, etc.) is analyzed [12, 22]. The paper [16] studies the optimal content placement in a cache network where requests are routed over fixed paths. Another important category of previous papers focus on the caching systems with multiple clients requesting caching services to one or multiple caching servers. For example, in [31], the authors study a system with one server and multiple clients to minimize the sum download and caching costs. In [27], FairRide, a strategy-proof caching scheme, is proposed to ensure caching fairness where multiple selfish users share a common storage. In [4], the bandwidth cost is minimized in a hierarchical tree caching topology. In [32], service cost is substantially reduced through predicting future requests and pre-caching potential requested files. In [33], the content placement problem in a distributed server network is addressed to improve uplink bandwidth utilization. In [10], the Square-Root caching scheme is proposed to minimize the expected search size (rather than a more general user utility discussed in Section 4 in this paper).

In mobile networks, in order to improve delay performance, backhaul efficiency, monetary cost, and energy efficiency, one widely advocated solution is to provide caching services at network edge, such as macrocell and small-cell base stations and WiFi access points [2, 23–26, 30, 34]. Another category of work investigates caching schemes in device-to-device (D2D) and ad hoc modes [7, 8, 15, 17–19]. Different from the scope of this paper, they do not characterize how popularity updates influence probabilistic caching. For example, the paper [18] discusses how to distribute cached contents through a network coding approach in a D2D network. In [7], the improvement in energy efficiency is studied in a cache-enabled D2D system. The prior papers more related to our work are [3, 9, 21], where optimal probabilistic caching schemes are proposed to optimize the system performance. However, our paper differs from [3, 9, 21] in several aspects. First, the optimization objectives are substantially different, leading to completely different analysis. Second, [3, 9, 21] are based on more specific assumptions (e.g., Poisson point distributed users and Zipf distributed popularity), while our analysis is based on more general assumptions. Finally, none of [3, 9, 21] considers re-caching policy when the popularity changes. To the best of our knowledge, this is the first paper to investigate probabilistic re-caching policy in mobile networks.

7 CONCLUSIONS

In this paper, we study the optimal probabilistic caching and re-caching policies in mobile networks. First, we propose the PFR policy in order to address the file re-caching problem as the file popularity changes. Through PFR, files cached (resp. not cached) are probabilistically opted out (resp. opted in) to achieve the updated popularity. We also prove that PFR satisfies five important properties: adaptability, robustness, computational efficiency, minimum file replacement, and insensitivity to minor variance. Second, we propose the OPCR policy, which jointly optimizes the file caching probabilities and the request limit. The optimization problem is non-convex in nature and cannot be solved with a standard method. In order to address this issue, we propose to decouple the decision variables to derive a globally optimal solution. Finally, prototype

[5]Note that in Fig. 4(e), there are $N = 50$ files in total. If 48 files are already cached, it is only meaningful to replace at most 2 files. If we force to replace 3 files, at least one file is opted out and then opted in again. That is why the number of replaced files is smaller when K becomes closer to 50.

based experiments and simulations are conducted to validate our proposed PFR and OPCR policies, demonstrating that the proposed policies significantly outperform non-optimal benchmark policies.

REFERENCES

[1] 2017. AllJoyn Framework. (2017). https://openconnectivity.org/developer/reference-implementation/alljoyn.

[2] Konstantin Avrachenkov, Jasper Goseling, and Berksan Serbetci. 2017. A Low-Complexity Approach to Distributed Cooperative Caching with Geographic Constraints. In *Proc. of ACM SIGMETRICS*. Champaign-Urbana, IL.

[3] Bartlomiej Blaszczyszyn and Anastasios Giovanidis. 2015. Optimal geographic caching in cellular networks. In *Proc. of IEEE ICC*. London, UK.

[4] Sem Borst, Varun Gupta, and Anwar Walid. 2010. Distributed Caching Algorithms for Content Distribution Networks. In *Proc. of IEEE INFOCOM*. San Diego, CA.

[5] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[6] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. 1999. Web caching and Zipf-like distributions: evidence and implications. In *Proc. of IEEE INFOCOM*. New York, NY.

[7] Binqiang Chen, Chenyang Yang, and Andreas F. Molisch. 2017. Cache-Enabled Device-to-Device Communications: Offloading Gain and Energy Cost. *IEEE Transactions on Wireless Communications* 16, 7 (Jul. 2017), 4519–4536.

[8] Zheng Chen and Marios Kountouris. 2016. D2D caching vs. small cell caching: Where to cache content in a wireless network?. In *Proc. of IEEE SPAWC*. Edinburgh, UK.

[9] Zheng Chen, Nikolaos Pappas, and Marios Kountouris. 2017. Probabilistic Caching in Wireless D2D Networks: Cache Hit Optimal Versus Throughput Optimal. *IEEE Communications Letters* 21, 3 (Mar. 2017), 584–587.

[10] Edith Cohen and Scott Shenker. 2002. Replication strategies in unstructured peer-to-peer networks. *ACM SIGCOMM Computer Communication Review* 32, 4 (Oct. 2002), 177–190.

[11] Aleksandr Ometov Ekaterina Olshannikova and, Yevgeni Koucheryavy, and Thomas Olsson. 2015. Visualizing Big Data with augmented and virtual reality: challenges and research agenda. *Journal of Big Data* 1, 2 (Dec. 2015), 1–27.

[12] Michele Garetto, Emilio Leonardi, and Stefano Traverso. 2015. Efficient Analysis of Caching Strategies Under Dynamic Content Popularity. In *Proc. of IEEE INFOCOM*. Hong Kong, China.

[13] Nicolas Gast and Benny van Houdt. 2015. Transient and Steady-state Regime of a Family of List-based Cache Replacement Algorithms. In *Proc. of ACM SIGMETRICS*. Portland, OR.

[14] Negin Golrezaei, Alexandros G. Dimakis, and Andreas F. Molisch. 2012. Wireless device-to-device communications with distributed caching. In *Proc. of IEEE ISIT*. Cambridge, MA.

[15] Maria Gregori, Jesss Gomez-Vilardebo, Javier Matamoros, and Deniz Gunduz. 2016. Wireless Content Caching for Small Cell and D2D Networks. *IEEE Journal on Selected Areas in Communications* 34, 5 (May 2016), 1222–1234.

[16] Stratis Ioannidis and Edmund Yeh. 2016. Adaptive Caching Networks with Optimality Guarantees. In *Proc. of ACM SIGMETRICS*. Juan-les-Pins, France.

[17] Mingyue Ji, Giuseppe Caire, and Andreas F. Molisch. 2016. Fundamental Limits of Caching in Wireless D2D Networks. *IEEE Transactions on Information Theory* 62, 2 (Feb. 2016), 849–869.

[18] Mingyue Ji, Giuseppe Caire, and Andreas F. Molisch. 2016. Wireless Device-to-Device Caching Networks: Basic Principles and System Performance. *IEEE Journal on Selected Areas in Communications* 34, 1 (Jan. 2016), 176–189.

[19] Jingjie Jiang, Shengkai Zhang, Bo Li, and Baochun Li. 2016. Maximized Cellular Traffic Offloading via Device-to-Device Content Sharing. *IEEE Journal on Selected Areas in Communications* 34, 1 (Jan. 2016), 82–91.

[20] Suoheng Li, Jie Xu, Mihaela van der Schaar, and Weiping Li. 2016. Popularity-Driven Content Caching. In *Proc. of IEEE INFOCOM*. San Francisco, CA.

[21] Derya Malak, Mazin Al-Shalash, and Jeffrey G. Andrews. 2016. Optimizing Content Caching to Maximize the Density of Successful Receptions in Device-to-Device Networking. *IEEE Transactions on Communications* 64, 10 (Oct. 2016), 4365–4380.

[22] Valentina Martina, Michele Garettoand, and Emilio Leonardi. 2014. A unified approach to the performance analysis of caching systems. In *Proc. of IEEE INFOCOM*. Toronto, Canada.

[23] Fidan Mehmeti and Thrasyvoulos Spyropoulos. 2017. Performance Modeling, Analysis, and Optimization of Delayed Mobile Data Offloading for Mobile Users. *IEEE/ACM Transactions on Networking* 25, 1 (Feb. 2017), 550–564.

[24] Francesco Pantisano, Mehdi Bennis, Walid Saad, and Merouane Debbah. 2014. Cache-aware user association in backhaul-constrained small cell networks. In *Proc. of IEEE WiOpt*. Hammamet, Tunisia.

[25] Konstantinos Poularakis, George Iosifidis, Antonios Argyriou, and Leandros Tassiulas. 2014. Video delivery over heterogeneous cellular networks: Optimizing cost and performance. In *Proc. of IEEE INFOCOM*. Toronto, Canada.

[26] Konstantinos Poularakis, George Iosifidis, Ioannis Pefkianakis, Leandros Tassiulas, and Martin May. 2016. Mobile Data Offloading Through Caching in Residential 802.11 Wireless Networks. *IEEE Transactions on Network and Service Management* 13, 1 (Mar. 2016), 71–84.

[27] Qifan Pu, Haoyuan Li, Matei Zaharia, Ali Ghodsi, and Ion Stoica. 2016. FairRide: Near-Optimal, Fair Cache Sharing. In *Proc. of USENIX NSDI*. Santa Clara, CA.

[28] Li Qiu and Guohong Cao. 2017. Popularity-Aware Caching Increases the Capacity of Wireless Networks. In *Proc. of IEEE INFOCOM*. Atlanta, GA.

[29] Lorenzo Saino, Ioannis Psaras, and George Pavlou. 2016. Understanding sharded caching systems. In *Proc. of IEEE INFOCOM*. San Francisco, CA.

[30] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G. Dimakis, Andreas F. Molisch, and Giuseppe Caire. 2013. FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers. *IEEE Transactions on Information Theory* 59, 12 (Dec. 2013), 8402–8413.

[31] Samta Shukla and Alhussein A. Abouzeid. 2017. Proactive Retention Aware Caching. In *Proc. of IEEE INFOCOM*. Atlanta, GA.

[32] John Tadrous and Atilla Eryilmaz. 2016. On Optimal Proactive Caching for Mobile Networks With Demand Uncertainties. *IEEE/ACM Transactions on Networking* 24, 5 (Oct. 2016), 2715–2727.

[33] Bo Tan and Laurent Massoulie. 2013. Optimal Content Placement for Peer-to-Peer Video-on-Demand Systems. *IEEE/ACM Transactions on Networking* 21, 2 (Apr. 2013), 566–579.

[34] Xiaofei Wang, Min Chen, Tarik Taleb, Adlen Ksentini, and Victor Leung. 2014. Cache in the air: exploiting content caching and delivery techniques for 5G systems. *IEEE Communications Magazine* 52, 2 (Feb. 2014), 131–139.