DATA CACHING IN AD HOC NETWORKS USING GAME-THEORETIC ANALYSIS

A Thesis by

Hoang Dang

Bachelor of Arts, Lakeland College, 2007

Submitted to the Department of Electrical Engineering and Computer Science and the
faculty of the Graduate School of
Wichita State University in partial fulfillment of
the requirements for the degree of
Master of Science

May 2011

DATA CACHING IN AD HOC NETWORKS USING GAME-THEORETIC ANALYSIS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Computer Science.

Bin Tang, Committee Chair

Pingfeng Wang, Outside Member

Neeraj Jaggi, Committee Member

ACKNOWLEDGEMENT

I would like to thank Dr Tang, my thesis advisor. I would not be able to complete this thesis without his support and guidance.

ABSTRACT

There have been researches that studied selfish data caching in ad hoc networks using game-theoretic analysis. However, due to the caching problem's theoretical root in classic facility location problem and k-median problem, most of the researches assume: 1) The data is initially outside of the network; 2) The caching cost is either a constant or not considered at all.

In reality, there are many applications, such as ad-hoc and sensor networks and peer to peer networks, in which data is initially collected or stored in the network and the caching cost depends on the network topology. This thesis addresses the problem of in-network data caching (referred to as in-caching problem) in multi-hop stationary ad hoc networks where the data is initially stored in the network and both caching and accessing costs are distance dependent. We first show that the problem is NP-hard. For selfish data caching game of the problem, we show that a pure Nash Equilibrium exists, in which a node will not deviate its caching strategy if others keep their own strategy. However, a Nash Equilibrium may not guarantee social optimal cost – due to the selfishness of each node, the price anarchy, which is the relative cost of the lack of cooperation among nodes, could be as large as *O(n)*, where *n* is a number of nodes in the network. Using an external incentive mechanism based upon a payment model, we show a Nash Equilibrium and social optimal can both be achieved simultaneously via extensive simulations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Ad hoc networks are multi-hop wireless networks consisting of small wireless computing devices with limited processing power, communication power, and storage capacity. The computing devices could be conventional computers such as PDAs or laptops, or embedded processors such as tiny, low-cost, and low-power sensor motes. Ad hoc networks are constructed mainly for the information sharing and task coordination among a group of people, without the support of any communication infrastructure. Each node in ad hoc networks not only participates in routing by forwarding data for other nodes, but also cooperate with each other to achieve a specific application goal. For example, in an ad hoc network established for spontaneous meeting, several authors can meet and coordinate to modify the same document (e.g., an article or a powerpoint slides) in a distributed fashion. Similarly, in a rescue and emergency operation, information sharing among wireless devices in an ad hoc networks enables coordination and cooperation of rescue personnel as well as gathering and distribution of information.

Caching has been proposed to be an effective technique to facilitate information access in ad hoc networks. Besides the traditional advantages brought by caching such as less data access latency, improved data reliability and fault tolerance, utilizing caching to optimize network performance of ad hoc networks is motivated by the following two aspects. First, the ad hoc networks are multi-hop networks. Thus, remote access of information typically occurs via multi-hop routing, wherein access latency can be particularly reduced by data caching. Second, ad hoc networks are generally resource constrained in terms of wireless bandwidth, memory capacity and battery energy of nodes. Data caching can help reduce communication cost among nodes, which results in conserving battery energy and minimizing bandwidth usage in ad hoc networks.

As such, many caching techniques have been developed recently to achieve good overall performance of the ad hoc network [5, 21, 20, 18, 4] (please refer to Chapter 2 for a comprehensive literature review). They are all cooperative caching techniques, wherein nodes follow carefully designed protocols to achieve overall good system performance. However, it is not always possible for nodes to cooperate with each other. Difference nodes could be under the control of different group and authorities, with different objectives. Even for the individual nodes under the same group domain, they could be autonomous, and as such, their staying in a group is solely due to their selfish goal of benefiting from the group. These are further aggravated by the fact that ad hoc networks are usually resource constraints, with limited wireless bandwidth, storage capacity, and processing power. Therefore, communicating nodes tend to be unwilling to contribute their own resource for the benefit of the entire network. For example, to achieve optimal data access in a network, some nodes will possibly cache data items which are most accessed by other nodes instead of themselves. Such *mistreatment* phenomenon [12] renders those nodes to break away from the group instead, so that they can store the data they access most. Therefore it is important to take into consideration such selfish behaviors when we want to improve the system performance.

On the other hand, several research work, among many others, has been performed to address the selfish data caching behavior using game-theoretic analysis in the context of distributed systems or Web environment [7, 2, 10, 3, 11]. Historically, data caching and the related cache placement problem have the theoretical root in facility location problem [6] and *k*-median problem [1]. Both problems concern how to place facilities in the network to satisfy the access demands from the client nodes with least amount of cost. Here the facility could be a delivery center, a distribution center, a transportation hub, a restaurant, or a cache node in a network. In the facility location problem, setting up a facility at a node incurs a certain fixed cost, and the goal is to minimize the sum of total access cost of clients and the setting-up costs of all facilities. The *k*-median problem is to minimize the total access cost under the constraint that at most *k* nodes can be selected as facilities, without considering setting-up costs. In both problems, the facilities to be set up are not initially in the network. As a result, most of work of selfish data caching assume that i) the data items are initially outside of the network (except for [7]), and ii) the caching cost is either a constant [2], does not exist [7, 3], or is only roughly categorized into only three difference values [11, 10].

While this assumption is valid in many scenarios, there are applications where the data items are instead in the network and caching cost depends on where the data is located inside the network. For example, in P2P networks, each peer initially has some data objects and shares them with other peers; in sensor networks, sensor nodes sense and generate data which are transmitted back to the base station for analysis or accessed by other sensor nodes in the network. In both cases, data are originally generated and stored in the network and the caching cost depends on where the data is cached from. Our model is geared towards such multi-hop ad hoc network, where the data items are initially stored at some nodes in the network (referred to as **source nodes**), and are subsequently cached by other nodes to better satisfy the access demands of the network. Unlike previous work, the efficiency of data caching scheme in in_Caching problem depends on not only the network topology and nodes' access patterns, but also the locations of the data item and its replica in the network.[1] We refer to our data caching problem as **in_Caching problem**.

In our model, there is one data item initially stored in a single source node, and there are multiple client nodes that wish to access the data item. Different nodes have different demands (referred to as **access frequencies**) to access the data item. They can either cache

[1] There exists, however, another data caching model wherein the data is initially in the network. We discuss it and elucidate its difference with our model in the Related Work.

the data from other existing cache node and store it in its local memory for future access, or just access the data from cache nodes whenever needed. The goal of the data caching problem is to determine a set of nodes in the network to cache the given data item, such that the total cost incurred in caching the data item and accessing the data item is minimized. While selfishness is considered, which we call selfish caching game, it endeavors to achieve optimal total cost based upon the selfish behavior of individual nodes.

We show that this problem itself is NP-hard. For the selfish caching game, we show that a pure Nash Equilibrium exists. Nash Equilibrium [14] is a solution concept of a game involving two or more players, in which no player has anything to gain by changing only its strategy unilaterally. It is important since it characterizes a “stable” outcome of the strategic interaction of several decision makers. In our problem, the strategy take by each node has two aspects: first, whether the node caches or not; second, if so, from where it caches, if not, from where it accesses the data. However, due to the selfish behavior of self-interested nodes, Nash Equilibrium may not guarantee system-wide performance; that is, the resultant social cost (or total cost) in Nash Equilibrium could be much larger than the social optimal cost (minimum total cost). Papadimitriou et al. [9, 16] quantify this as **price of anarchy**, which is the ratio of the social cost of the worst possible Nash equilibrium to the cost of the social optimal solution. We show that in our constructed Nash Equilibrium, the price of anarchy could be as large as $O(n)$, where n is number of nodes in the network. More importantly, we endeavor to show that that social optimal and Nash Equilibrium can be achieved simultaneously (i.e., with the price of anarchy as $O(1)$). We design an external incentive mechanism based upon a payment model, in which nodes benefit from other nodes' being caches offer some payment to such nodes.

Thesis Organization. The rest of the thesis is organized as follows. Chapter 2 reviews both cooperative and selfish data caching in the literature. In Chapter 3, we introduce in-networking data caching network model and formulate the problem, and show that it is NP-hard. In Chapter 4 we formalize the selfish data caching game of the problem and demonstrate by a centralized construction that a pure Nash Equilibrium exists. Chapter 5 presents our payment model which achieves the optimal social cost as well as aN ash E quilibrium. I n Chapter 7, we conclude the paper and poi nt out s ome future work.

Chapter 2

Related Work

2.1 Cooperative Caching in Ad Hoc Networks

There are lot of research designing distributed caching algorithms in ad hoc networks. Hara and Madria [5] are among the first to propose replica allocation methods in ad hoc networks, by taking into account the access frequency from mobile hosts to each data item and the status of the network connection. Yin and Cao [20] design and evaluate three simple distributed caching techniques, viz., **CacheData** which caches the passing-by data item, **CachePath** which caches the path to the nearest cache of the passing-by data item, and **HybridCache** which caches the data item if its size is small enough, else caches the path to the data. Fiore et al. [4] design a cooperative caching scheme to create a content diversity in ad hoc networks, so that a requesting user likely finds the desired information nearby. Zhao et al. [21] propose a novel asymmetric cooperative cache approach, where the data requests are transmitted to the cache layer on every node, but the data replies are only transmitted to the cache layer at the intermediate nodes that need to cache the data.

Ko and Rubenstein [8] propose a distributed protocol that palaces replicated resources in a network such that the distance between identical copies of the same resource is large and each node is "close" to some copy of any resource. They study it by coloring each node, where each color is a replica the node is assigned. It proves the network can converge to a stable state following such protocol. In our previous work [18], we present a polynomial-time centralized approximation algorithm to replicate data, which reduces the total data access delay at least half of that obtained from the optimal solution. We also show a distributed caching technique derived from the centralized approximation algorithm.

Our data caching problem is closely related to the rent-or-buy problem [17], a special case of the connected facility location problem [13, 17, 15] (please refer to Chapter 3 for the detailed comparison of these two problems). Swamy et al. [17] give a 5-approximation algorithm. Nuggehalli et al. [15] study the same problem in the context of energy-efficient caching strategies in ad hoc networks. They provide a distributed solution that is within a factor of 6 of the optimal solution.

However, all of above work do not take into consideration of selfishness of the network node, which is the topic of this work.

2.2 Selfish Caching in Ad Hoc Networks and Distributed Systems

Chun et al. [2] are among the first to propose to study selfish caching in distributed systems using a game-theoretic approach. They consider one data object which is outside of the network. When a node decides to cache the data object, it assumes that the node always gets this data from outside of the network, which incurs a constant caching cost. A node either caches the data object in its local memory or accesses it from another node storing the object, depending on which costs less. They show that there exists a pure strategy Nash Equilibrium based on above model. However, the total social cost can not achieve optimum due to selfish behavior of players. They propose a payment model, in which each node bids for having an object replicated at another node, and show that both social optimal and Nash Equilibrium can be achieved.

Laoutaris et al. [10] study distributed selfish replication and caching of multiple objects. In their model, the set of objects are also not in the network initially and the caching cost is not considered. Moreover, the distances between nodes are not factored in when playing the game. Rather, it assumes that for each node, accessing an object from its local cache always costs t_l, from another cache node t_r, and from the origin server always t_s, with $t_l \leq t_r \leq t_s$.

The contributions of their paper are two fold: a) they consider memory capacity of each node since multiple objects are involved; b) the Nash Equilibrium object placement strategies are implemented in a distributed manner. The authors further extend their work by identifying and and investigating the causes of mistreatment [11, 12].

Our work considers one in-network data item in a multi-hop ad hoc network, wherein both accessing cost and caching cost not only exist, but also are topology dependent.

Chapter 3

Network Model and Problem Formulation

In this section, we first present the network model including the cost model, formulate the problem, and discuss its difference with closely related rent-or-buy problem. We then present an important entity, called cache tree, in the problem. Finally we prove that the problem is NP-hard.

Network Model and Notations. We model the ad hoc network as a connected general graph, $G(V, E)$, where $V = \{1, 2, 3, ..., n = |V|\}$ is set of nodes and E is set of edges. Two nodes are connected by an edge if they are within the transmission range of each other and thus can communicate directly. There is a single data item D in the network, which is stored in its original source node $S \in V$. D is requested by the nodes in the network – each node has its own access frequency towards the data; the access frequency of node $\boldsymbol{i}$ is $a_i \in R^+$. Let d_{ij} be the shortest distance (in terms of number of hops) between node $\boldsymbol{i}$ and node $\boldsymbol{j}$. For a set of nodes $X \subseteq V$, let $d(i, X) = \min_{j \in X} d_{ij}$ be the shortest distance from $\boldsymbol{i}$ to some node in X, and the **metric closure** of X is defined as the complete graph upon X, wherein an edge between two nodes in X is a shortest path between them in the original network graph $G(V, E)$. Let $mst\ mc(X)$ denote the cost of a minimum spanning tree of the metric closure upon X.

Cost Model. For a given set of cache nodes $M \subseteq V$ (source node S is also considered as a cache node), a non-cache node $\boldsymbol{i}$ always accesses D from the closest cache node among M; its **access cost** is $a_i d(i, M)$. The **total access cost** is therefore $\sum_{i \in V} a_i \times d(i, M)$. For the set of cache nodes M, any cache node $\boldsymbol{i} \in M$ caches the data from another cache node in M, say $\boldsymbol{j}$, by following a shortest path between these two nodes (we call such shortest path the **caching path**); and its **caching cost** is $\gamma \times d_{ij}$. Here γ, referred to as **caching coefficient**, is a constant that indicates the relative weight of caching cost compared to access cost. We emphasize that $\boldsymbol{j}$, however, might not be the closest cache node to $\boldsymbol{i}$ in M, due to the fact that caching has time sequence in which a later cache node in M, say k, might be closer to $\boldsymbol{i}$ than $\boldsymbol{j}$ is. The **total caching cost** of all the nodes in M is therefore $\gamma \times mst\_mc(M)$.

Let $\tau(M)$ denote the **total cost in the network** with a set of cache nodes M (source node $S \in M$), we have:

$$\tau(M) = \sum_{i \in V} a_i \times d(i, M) + \gamma \times mst\_mc(M) \tag{3.1}$$

In above equation, the two terms on the right hand side represent total access cost and total caching cost in the network respectively. Notice that, by varying γ, we can model different network scenarios and requirements.

Problem Formulation of in_Caching Problem. Given a network graph $G(V, E)$, one data item D and its source node S, and access frequencies of all client nodes, the objective is to select a set of cache nodes $M \subseteq V$, such that the total cost in the network given by Equation 3.1 is minimized, i.e.,

$$\min_{M} \tau(M) \tag{3.2}$$

Let τ^{opt} denote the optimal cost, $\tau^{opt} = \min_M \tau(M)$. Let C^{opt} denote the set of cache nodes (including S) in the optimal solution, $C^{opt} = \text{argmin}_M \tau(M)$.

Discussion. In_Caching problem is closely related to the well studied rent-or-buy problem [17]. The rent-or-buy problem is defined as follows. One facility is already open, along with a set of locations at which facilities can be further built. Connecting the facilities incurs a cost which is proportional to the weight of the Steiner tree [19] connecting all the facilities, and each client accesses its closest facility. The objective is to select the locations to build facilities and connect them by a Steiner tree, to minimize the total access cost and connecting cost. The rent-or-buy problem is known to be NP-hard [17].[1] In our problem setting, the opened facility represents the source node S, the set of opened facilities corresponds to the set of cache nodes.

The difference between in_Caching problem and rent-or-buy problem lies in the caching models. In rent-or-buy problem, the total caching cost is the cost of the Steiner tree connecting all the facilities; while in in_Caching, the total caching cost is the cost of minimum spanning tree of the metric closure upon all the cache nodes. Essentially, in Steiner tree model, an intermediate node between two cache nodes can make a copy of the data, which is then cached by other nodes; while in our model, however, a cache node caches the data from an already existing cache node in the network. While Steiner tree has been used widely to model communication connectivity among nodes, it is not suitable to model data caching in ad hoc networks, for the following two reasons.

- The ad hoc nodes such as sensor nodes and laptops have very limited storage capacity. By requiring the intermediate nodes between two cache nodes to make a copy of the data, steiner tree caching model further aggravates the situation of limited storage capacity for ad hoc networks.

- Even though Steiner tree is the optimal tree to connect all the cache nodes, in many applications such as wireless P2P networks [21], two participating nodes (client node and server node of the shared file) do not want all other peers in their communication path to make and store a copy of the shared file, for reasons like ownership, trust, and privacy.

Cache Tree. The caching paths give the parent-child relationship of all the cache nodes,

[1] If everything else being the same, except that there is no open facility initially, the problem is called connected facility location, which is also NP-hard [13, 17, 15].

indicating from which parent cache node that each child cache node directly caches the data item from. Such relationships form a tree, referred to as the **cache tree**, which is rooted at source node S. Below we give definitions related to the cache tree.[2]

Definition 1 (Parent Cache and Child Cache.) In cache tree, cache node $\boldsymbol{i}$ is the **parent cache** of cache node $\boldsymbol{j}$, denoted as $P(\boldsymbol{i})$, if $\boldsymbol{j}$ directly fetches the data from $\boldsymbol{i}$. $\boldsymbol{j}$ is a **child cache** of $\boldsymbol{i}$. ✷

Definition 2 (Ancestor Cache and Descendant Cache.) Cache node $\boldsymbol{i}$ is an **ancestor cache** of cache node $\boldsymbol{j}$ ($\boldsymbol{i} = \boldsymbol{j}$) if $\boldsymbol{i}$ is on the unique path from S to $\boldsymbol{j}$ (including S but not $\boldsymbol{j}$) in the cache tree. That is, $\boldsymbol{j}$ directly or indirectly fetches the data from $\boldsymbol{i}$. $\boldsymbol{j}$ is a **descendant cache** of $\boldsymbol{i}$. ✷

Definition 3 (Descendant Set.) The **descendant set** of cache node $\boldsymbol{i}$, denoted as $D(\boldsymbol{i})$, is the set of $\boldsymbol{i}$'s descendant caches. In other words, $D(\boldsymbol{i})$ is all the nodes in the subtree rooted at $\boldsymbol{i}$ (including node $\boldsymbol{i}$). For example, the descendant set $D(S)$ of source node S is the entire cache tree. ✷

Definition 4 (Selected Time of Cache Node.) Cache tree mandates the timeline of cache node selection, i.e the cache nodes in the cache tree can be ordered using the time at which they are selected as cache nodes. We use $t(\boldsymbol{i})$ to indicate the time sequence at which $\boldsymbol{i}$ is selected as cache node and assume $t(S) = 0$. For any cache node $\boldsymbol{i}$, we have $t(D(\boldsymbol{i})) > t(\boldsymbol{i})$.
✷

Theorem 1 *The in_Caching problem is NP-hard.*

Proof: The in_Caching problem can be proved to be NP-hard via a reduction from the facility location problem (FLP) [6]. The FLP is similar to in_Caching problem with two

[2] Note that the cache tree is a "logical" tree, because its corresponding caching paths do not necessarily form a tree, with possible overlapping edges or even loops.

differences: i) the data item is initially outside of the network and ii) the caching cost is a constant. In in_caching problem, the caching cost of a cache node depends on its distance to another cache node from which it caches the data, thus is not the same for all the cache nodes. Therefore FLP is a special case of in_Caching when caching cost is a constant, which shows in_Caching problem is also NP-hard. ∎

Chapter 4

Basic Selfish Caching Game

In selfish caching game, however, whether a node caches the data itself or accesses the data from other cache nodes depends on which costs less, not according to optimal solution. Below we first discuss the cost model in the selfish caching game, which is different from that of in_Caching problem discussed in Chapter 3. Then we show that a pure Nash Equilibrium exists. Finally we discuss the performance of Nash Equilibrium in terms of the price of anarchy, which quantifies the cost of the lack of cooperation among nodes.

4.1 Cost Model

For each non-source node, it either caches the data in its local memory (cache node) or accesses the data from others (non-cache node). For node ***i***, its **access cost** and **caching cost** are denoted as α_i or β_i respectively.

Access Cost. For a non-cache node ***i***, once all the cache nodes are selected and have data cached in their local memories, ***i*** accesses D from its closest cache node (including the source node). Assuming k is ***i***'s closest cache node storing D, then ***i***'s access cost α_i is $a_i d_{ik}$.

Caching Cost. When a cache node ***i*** decides to cache data from other existing cache nodes, it goes to the closest one, say k, to fetch the data and cache it into its local memory. Thus ***i***'s caching cost β_i is also proportional to the shortest distance to cache node k and $\beta_i = \gamma d_{ik}$. As in Chapter 3, γ is a constant indicating the relative weight of caching cost to access cost.

Total Cost in Nash Equilibrium. In a caching game, each node is either a cache node or non-cache node; whether a node ***i*** is a cache node or not depends on a_i and γ: if $a_i \geq \gamma$, ***i*** is a cache node and caches the data from its nearest existing cache node[1]; if $a_i < \gamma$, it is a non-cache node and accesses the data from the nearest one among all the cache nodes.

Let the **cost of node *i*** of requesting D be τ_i, then τ_i equals either α_i or β_i. Let τ^N denote the total cost of the network in Nash Equilibrium, $\tau^N = \sum_{i \in V} \tau_i$. Let C^N denote the set of cache nodes (including S) in a Nash Equilibrium, $C^N = \{i | a_i \geq \gamma, 1 \leq i \leq n\} \cup \{S\}$. Let $m = |C^N|$.

Caching Strategy. The caching strategy of node i, denoted as C_i, includes the following. First, it decides whether it is a cache node or not. Second, if yes, it decides from which cache node (its parent cache) it fetches the data and caches in its local memory; if no, it decides from which cache node it accesses the data item D. More formally, $C_i = (n_i, P(i), c_i)$, where $n_i \in \{yes, no\}$ indicates if i is a cache or not, $P(i)$ is the parent cache node of i if i is a cache node, otherwise i accesses D from its closest cache node c_i. Let SP denote the **strategy profile** of the game, $SP = \{C_1, C_2, ..., C_n\}$ shows the global cache placement and data access in the entire network.

Before we present the algorithm that achieves Nash Equilibrium, we first show a property of the Nash Equilibrium achieved in the caching game, which says that a cache node in Nash Equilibrium is still a cache node in the optimal solution.

Lemma 1 $C^N \subseteq C^{opt}$.

Proof: We prove it by contradiction. Assume node $i \in C^N$ is not a cache node in the optimal solution, i.e $i \notin C^{opt}$. We have that $a_i \geq \gamma$. In an optimal solution (there could be multiple optimal solutions), assume i accesses another cache node l for the data item D. To further reduce the total cost, i can cache the data from l. This further reduces the total cost of the entire network by $(a_i - \gamma) \times d_{il}$, contradicting that it is an optimal solution. Therefore $C^N \subseteq C^{opt}$.

∎

[1] Here we emphasize that this cache node is the nearest one to i among the existing cache nodes at the time when i becomes a cache node.

We use C^A to denote the set of non-cache nodes in Nash Equilibrium in the basic selfish caching game that are cache nodes in the optimal solution, i.e., $C^A = C^{opt} - C^N$.

4.2 Nash Equilibrium Construction

Below, we first present the algorithm leading to a Nash Equilibrium. Then we present some observation from the algorithm, which serves as the basis of Nash Equilibrium proof in Theorem 2 at the end of this subsection. For the clarity of the presentation, we call a cache node before it caches data a **potential cache node**.

Nash Equilibrium Construction. The algorithm takes place in iterations. In each iteration, a potential cache node is selected as cache node and caches data from an already existing cache node, the caching path being a shortest path between them. There are $m - 1$ non-source cache nodes, so the algorithm stops after $m - 1$ iterations.

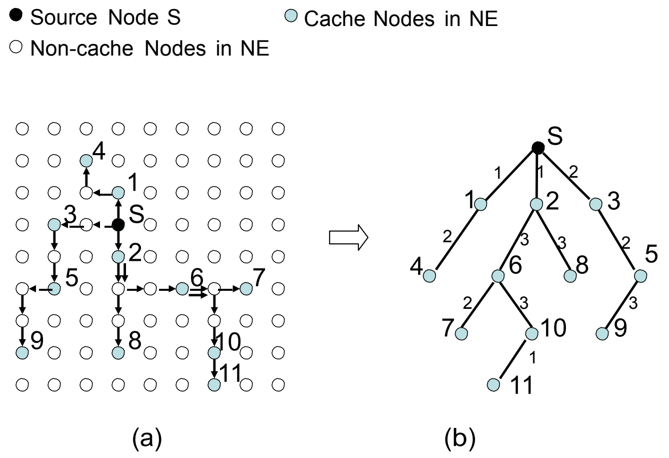

Figure 4.1: Nash Equilibrium construction in a grid-like ad hoc network. (a) shows the caching paths. The ID of a cache node is the time sequence at which it gets a data copy from its parent cache following the direction of the arrowed edge. Note that some edges are used multiple times, indicated by the double arrowed edges. (b) shows the cache tree in the Nash Equilibrium, with nodes as the cache nodes and the number on the edge the cost of the shortest path between each pair of parent-child cache nodes.

Minimum Caching Cost Algorithm for Nash Equilibrium.

(1) Start with the source node S, find the potential cache node with the minimum shortest distance to the source node (when there is a tie, the potential cache node with smaller ID is selected). It is a new cache node. Move a copy of the data from the source node to the new cache node along the shortest path between them.

(2) **while** (there is still un-cached potential cache node)

Among all the shortest path linking any existing cache node to any potential cache node, find the one with the minimum cost. When there is a tie of the parent cache and/or child cache, choose the one with smaller ID. Move a copy of the data from the selected parent cache node to the selected child cache node along the shortest path between them.

(3) For each of the non-cache nodes, it accesses the data from its nearest cache node.

Above is essentially the minimum spanning tree algorithm upon the metric closure of all the cache nodes. Furthermore, above algorithm results in a unique minimum spanning tree because all the ties are broken following the unique IDs of nodes. Note also that the algorithm is in the same line as the total caching cost modeled in Chapter 3.

EXAMPLE 1 Figure 4.1 (a) shows such Nash Equilibrium construction for a grid-like network topology, where each node can only communicate directly with its (at most four) neighbors.[2] All the caching paths are shown as the arrowed edges, the direction of which indicating the movement of the data item from parent cache node to child cache node. For the ease of presentation, the ID of each node in the figure now indicates the iteration (time sequence) at which the cache node gets a data copy from its parent cache following the caching path. ✷

Cache Tree in Nash Equilibrium. Figure 4.1 (b) shows the cache tree corresponding to the caching paths in Figure 4.1 (a). The nodes of the cache tree are the set of cache

[2] Note that the grid-like topology is only for the purpose of ease of presentation, above Nash Equilibrium construction is applicable to any topologies.

nodes. Each edge represents a parent-child cache node relationship. The number on the edge indicates the cost of the shortest path between each pair of parent-child cache nodes. It is noted that corresponding shortest paths do not comprise of tree, since the cache path between 2 and 6 and the cache path between 2 and 8 overlaps partially, as shown in Figure 4.1 (a)

<u>Selection of Parent Cache.</u> In the caching game, when i is selected to become a cache node, it chooses the nearest *existing* cache node, $P(i)$, as its parent cache and fetches the data from $P(i)$. However, a later selected cache node j could be closer to i than $P(i)$, causing cache node i to deviate and cache the data from j instead. Surprisingly, we show that for all the cache nodes selected after i, only nodes in $D(i)$ can possibly be closer to i than $P(i)$. More generally, Lemma 2 below shows that if a cache node j is not a descendant cache of i, then i does not have incentive to deviate to cache from j.

Lemma 2 *For two cache nodes i and j in a cache tree, if $j \not\in D(i)$, then $d_{iP(i)} \leq d_{ij}$, which means that i does not have incentive to deviate from its parent cache $P(i)$ to have j as its parent cache.*

Proof: When $t(i) > t(j)$ and $j \not\in D(i)$, we have $d_{iP(i)} \leq d_{ij}$. This is because following minimum caching cost algorithm, at the iteration when i is selected as the cache node and caches data from $P(i)$, $iP(i)$ is the minimum shortest path among all the shortest paths connecting any cache node to any potential cache node, i.e., $d_{iP(i)} \leq d_{ij}$.

When $t(i) < t(j)$ and $j \not\in D(i)$, as shown in Figure 4.2, along the path from j to S (including j and S), there must exist one cache node, say k, with $t(P(k)) < t(i) < t(k)$ (note $P(k)$ and $P(i)$ could be the same node). We have $d_{iP(i)} \leq d_{kP(k)}$ following the minimum caching cost algorithm. Using similar argument, we have $d_{kP(k)} \leq d_{ji}$. Therefore $d_{iP(i)} \leq d_{ij}$, indicating that i does not have incentive to cache from j. ∎

For example, in Figure 4.1 (a), since $t(6) < t(9)$ and node $9 \not\in D(6)$, node 6 will not deviate from its parent cache node 2 to cache from node 9. This can be confirmed by that the distance between node 6 and node 2 is 3 hops, which is less than distance between node 6 and node 9, which is 7 hops.

Observation 1 *In our in-network data caching model, we observe that an ancestor node can not access or cache data from its descendant cache nodes. Because otherwise, it violates the intrinsic ancestor-descendant relationship between ancestor and descendant nodes. This observation, together with Lemma 2, lead to below theorem about the Nash Equilibrium construction.*

Theorem 2 *The minimum caching cost algorithm reaches Nash Equilibrium.*

Proof: To prove that Nash Equilibrium exists, we need to show each node does not unilaterally deviate from its caching strategy. That is, the caching node keeps its parent cache node, while non-cache node accesses data from the same cache node. For any non-cache node $\boldsymbol{i}$, it will not deviate to be a caching node since its access cost is less than its caching cost, due to $a_i < \gamma$; it will not access from another cache node since it accesses the data from closest cache node.

For any cache node $\boldsymbol{i}$, it will not deviate to be a non-cache node because $a_i \geq \gamma$. Below we show it will not deviate from its parent cache node $P(i)$ to any other cache node, say $\boldsymbol{j}$. This is true when $\boldsymbol{j} \not\in D(\boldsymbol{i})$, because from Lemma 2, $\boldsymbol{i}$ will not deviate to have $\boldsymbol{j}$ as its parent cache node. When $\boldsymbol{j} \in D(\boldsymbol{i})$, it does not deviate either since it is prohibited that ancestor node accesses or caches data from its descendant cache.

No node deviates unilaterally. We conclude that the minimum caching cost algorithm reaches Nash Equilibrium.

∎

4.3 Price of Anarchy (PoA)

Price of Anarchy is defined as the ratio between the social cost of the worst possible Nash equilibrium to the cost of the social optimal solution [9, 16]. Below we discuss the PoA in our constructed Nash Equilibrium.

Lemma 3 *If $a_i \geq \gamma$ for all $i \in V$, the PoA of the data caching game is $O(1)$.*

Proof: In this case, all the nodes in the network are cache nodes, and the total cost is $\tau = \sum_{i \in V} \gamma d_{iP(i)}$. The metric closure upon all the cache nodes is the same as $G(V, E)$. The minimum caching cost algorithm upon the metric closure is essentially the same as the minimum spanning tree algorithm upon $G(V, E)$. Thus both yield the same cost, PoA of the game is $O(1)$. ∎

Lemma 4 *If $\exists\, i \in V$, such that $a_i < \gamma$, the PoA of the game can be $O(n)$.*

Proof: We prove this by showing an example depicted in Figure 4.3. In this example, nodes $\{2, 3, ..., n-1, n\}$ are all non-cache nodes, with $a_2 = a_3 = a_n = a < \gamma$. The distance between node 2 and source node S is 1 and the distance between nodes $3, 4, ..., n-1, n$ to node 2 is 0. The Nash Equilibrium is that all the non-cache nodes access the data from S, giving total cost $\tau^N = a \times (n-1)$. However, the social optimal solution is that node 2 caches the data while nodes $3, 4, ..., n-1, n$ access it from node 2, yielding optimal total cost $\tau^{opt} = \gamma$. Therefore PoA = $\frac{\tau^N}{\tau^{opt}} = \frac{(n-1)a}{\gamma}$, which is $O(n)$. ∎

Lemma 4 shows that due to the non-cooperation among selfish nodes, the social optimal is not achieved in the Nash Equilibrium. Below, we present a payment-based mechanism wherein some non-cache nodes are made payment by other nodes. We show our proposed mechanism can achieve both Nash Equilibrium and social optimal.

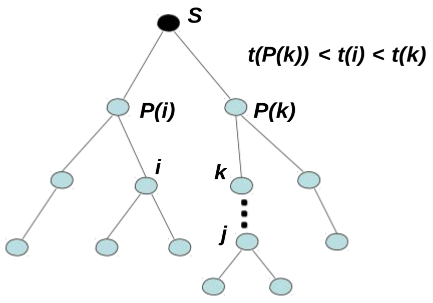

Figure 4.2: In a cache tree, if $t(j) > t(i)$ and $j \not\subseteq D(i)$, then $d_{iP(i)} \leq d_{ij}$, which means i does not have incentive to deviate from $P(i)$ to have cache node j as its parent cache.

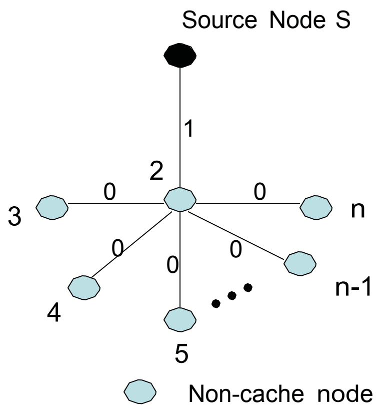

Figure 4.3: An example showing $PoA = O(n)$ in the selfish caching game.

Chapter 5

Payment-based Selfish Caching Game With Optimal Solution

In this section, we design a payment-based mechanism wherein a Nash Equilibrium and a social optimal solution are achieved simultaneously (we call it the **NE-based optimal solution** for the rest of the paper). Recall C^A is the set of non-cache nodes in basic game Nash Equilibrium that are cache nodes in the optimal solution. The idea of the payment-based mechanism is to incentivize the nodes in C^A to cache data so that the optimal solution can be obtained and meanwhile they do not deviate unilaterally. Specifically, we need to answer the following three questions:

- For node $\boldsymbol{i} \in C^A$, being a selfish player, at least how much incentive it needs to get to stay as a cache node in the optimal solution?
- Among all the nodes in V, which nodes should incentivize node $\boldsymbol{i}$ by offering some payments to $\boldsymbol{i}$?
- How much payment each should offer?

Before we answer these questions, we consider again the grid-like ad hoc network shown in Figure 4.1 (a), and study the data caching in the NE-based optimal solution.

Cache Tree in Optimal Solution. Figure 5.1 (a) shows the set of cache nodes and cache paths in the NE-based optimal solution for the grid-like ad hoc network shown in Figure 4.1 (a). It shows that $C^A = \{1', 2', 3', 4'\}$. For the nodes in C^N (recall that the set of cache nodes in basic game Nash Equilibrium are still the cache nodes in the optimal solution), we keep their IDs the same as in Figure 4.1 (a) for simplicity. Therefore all the node IDs in Figure 5.1 (a) no longer reflect the time sequence each cache node caches the data from its parent cache. Figure 5.1 (b) shows the corresponding cache tree. Below we use it to illustrate our observations for the NE-based optimal solution and to answer above three questions.

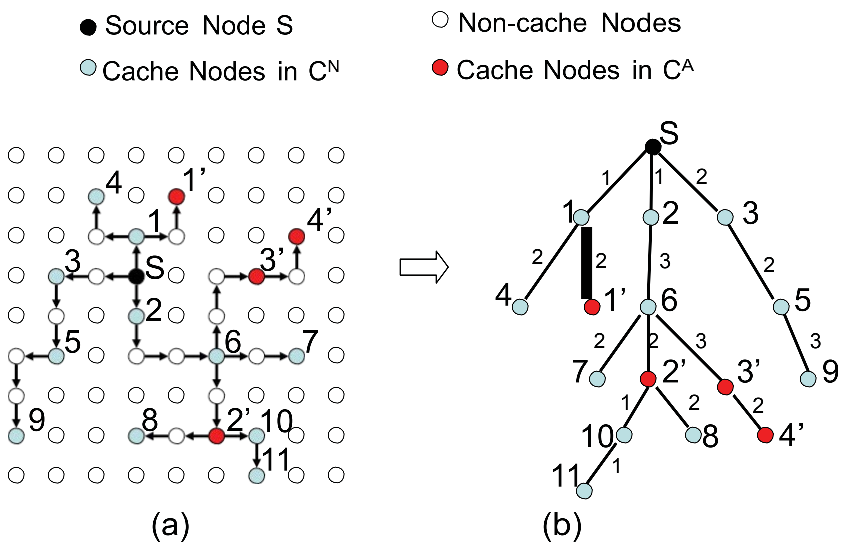

Figure 5.1: Illustration of the NE-based optimal solution. (a) shows the cache nodes and caching paths. (b) shows the corresponding cache tree.

Lemma 5 *At least* $(\gamma - a_i) \times d_{iP(i)}$ *amount of payment is needed to incentivize* $\boldsymbol{i}$ *to stay as a cache node, where* $P(\boldsymbol{i})$ *is the parent cache of node* $\boldsymbol{i}$ *in the optimal solution. Otherwise,* $\boldsymbol{i}$ *will deviate to access the data from* $P(\boldsymbol{i})$.

Proof: If node $\boldsymbol{i}$ decides to deviate from being a cache node, it will access the data from the nearest cache node in $C^{opt} - \{\boldsymbol{i}\}$. From Observation 1, node $\boldsymbol{i}$ can not access data from its descendant cache nodes $D(\boldsymbol{i})$. From Lemma 2, for two cache nodes $\boldsymbol{i}$ and $\boldsymbol{j}$ in a cache tree, if $\boldsymbol{j} \notin D(\boldsymbol{i})$, then $d_{iP(i)} \leq d_{\boldsymbol{ij}}$. Therefore, the nearest cache node in $C^{opt} - \{\boldsymbol{i}\}$ for $\boldsymbol{i}$ to access is $P(\boldsymbol{i})$. To stay as a cache node, it incurs caching cost $\gamma \times d_{iP(i)}$. Therefore, at least $(\gamma - a_i) \times d_{iP(i)}$ amount of payment is needed as incentive for $\boldsymbol{i}$ to stay as a cache node. ∎

For the second question, we observe that the nodes which are willing to pay to incentivize cache node $\boldsymbol{i} \in C^A$ are in two categories. First, they are those non-cache nodes whose access cost will possibly increase if $\boldsymbol{i}$ accesses the data instead of caching it. Obviously, they are the non-cache nodes that access data from $\boldsymbol{i}$. Second, they are the cache nodes whose caching cost will possibly increase if $\boldsymbol{i}$ accesses the data instead of caching it. Such set of cache nodes can be easily found by running the minimum spanning tree algorithm upon the metric closure of all the nodes in $C^{opt} - \{\boldsymbol{i}\}$. Lemma 6 below demonstrates such cache nodes.

Lemma 6 *If $\boldsymbol{i} \in C^A$ decides to access the data instead of caching it from $P(\boldsymbol{i})$, among all the cache node $\boldsymbol{j} \in C^{opt}$, only the cache node in $D(\boldsymbol{i})$ could possibly incur increased caching cost.*

Proof: It suffices to show that for $\boldsymbol{j} \not\in D(\boldsymbol{i})$, if $\boldsymbol{i}$ decides not to cache, $\boldsymbol{j}$'s parent cache is still $P(\boldsymbol{j})$, and its caching cost does not change.

If $t(\boldsymbol{j}) < t(\boldsymbol{i})$, $\boldsymbol{j}$'s caching path is not affected because due to the uniqueness of the caching tree, $\boldsymbol{j}$ still caches before $\boldsymbol{i}$ does. If $t(\boldsymbol{j}) > t(\boldsymbol{i})$, since $\boldsymbol{j} \not\in D(\boldsymbol{i})$, by way of contradiction, if $\boldsymbol{j}$ changes its parent cache from $P(\boldsymbol{j})$ to another cache node, say k, as the result of minimum spanning tree algorithm upon the metric closure of all the nodes in $C^{opt} - \{\boldsymbol{i}\}$, this results that the cache tree of all nodes in C^{opt} is not minimum, which contradicts with the fact that such tree is minimum spanning tree upon the metric closure of all the nodes in C^{opt}.

Second, we show $\boldsymbol{j} \in D(\boldsymbol{i})$'s caching cost will possibly increases if $\boldsymbol{i}$ decides not to cache.

■

Below we answer the third question that how much amount payment each of the nodes which benefit from $\boldsymbol{i}$'s being cache node is willing to pay $\boldsymbol{i}$. We give the following definitions. For each cache node $\boldsymbol{i} \in C^A$, we denote the set of non-cache nodes accessing data from $\boldsymbol{i}$ as $\Theta_{\boldsymbol{i}}$.

Definition 5 (Benefit of cache node $\boldsymbol{i} \in C^A$ to a non-cache node $\boldsymbol{j} \in \Theta_{\boldsymbol{i}}$.) We define the benefit of cache node $\boldsymbol{i} \in C^A$ to non-cache node $\boldsymbol{j} \in \Theta_{\boldsymbol{i}}$ ($a_{\boldsymbol{j}} < \gamma$), denoted as $\psi_{\boldsymbol{j}}$, as the minimum extra cost incurred to $\boldsymbol{j}$ if $\boldsymbol{i}$ decides not to cache. $\boldsymbol{j}$ therefore accesses the data from the closest cache node in $C^{opt} - \{\boldsymbol{i}\}$. Formally, $\psi_{\boldsymbol{j}} = a_{\boldsymbol{j}} \times (\min_{l \in (C^{opt} - \{i\})} d_{\boldsymbol{j}l} - d_{\boldsymbol{j}i})$. ✷

Definition 6 (Benefit of cache node $\boldsymbol{i} \in C^A$ to a cache node $k \in D(\boldsymbol{i})$.) We define the benefit of cache node $\boldsymbol{i} \in C^A$ to a cache node $k \in D(\boldsymbol{i})$ ($a_k \geq \gamma$), denoted as φ_k, as the minimum extra cost incurred to k if $\boldsymbol{i}$ decides not to cache. k has to cache the data from another cache node in $C^{opt} - \{\boldsymbol{i}\}$. Formally, $\varphi_k = \gamma \times (d_{kP'(k)} - d_{kP(k)})$, where $P'(k)$ is the parent cache of k following the minimum spanning tree algorithm over the metric closure of the set of cache nodes $C^{opt} - \{\boldsymbol{i}\}$, and $P(k)$ is the parent cache of k when the set of cache nodes are C^{opt}. ✷

Definition 7 (Benefit and average benefit of cache node $\boldsymbol{i} \in C^A$ to the entire network.) Now, the benefit of cache node $\boldsymbol{i} \in C^A$ to the entire network, denoted as Ξ_i, is the increase of the total cost of the entire network if $\boldsymbol{i}$ decides not to cache the data from $P(\boldsymbol{i})$, but to access it from $P(\boldsymbol{i})$. Formally, $NB_i = \sum_{\boldsymbol{j} \in \Theta_i} \psi_{\boldsymbol{j}} + \sum_{k \in D(i)} \varphi_k - (\gamma - a_i) \times d_{iP(i)}$. The average benefit of node $\boldsymbol{i}$ to the entire network, denoted as nb_i, is the average cost saving for all the nodes in Θ_i and $D(\boldsymbol{i})$, plus itself, if node $\boldsymbol{i}$ decides to cache. That it, $nb_i = NB_i / (|\Theta_i| + |D(\boldsymbol{i})| + 1)$.
✷

Lemma 7 $nb_i \geq 0$.

Proof: By way of contradiction, assume $nb_i < 0$ in the optimal solution. That is, $\sum_{\boldsymbol{j} \in \Theta_i} \psi_{\boldsymbol{j}} + \sum_{k \in D(i)} \varphi_k - (\gamma - a_i) \times d_{iP(i)} < 0$. Consider the new caching solution where $\boldsymbol{i}$ decides not to cache and thus each node in Θ_i and $D(\boldsymbol{i})$ chooses its next best strategy, and all other nodes in $C^{opt} - \{\boldsymbol{i}\}$ are still cache nodes. With Lemma 6, it is easy to see that the optimal cost minus the cost of the new caching solution is equal to:

$$
\begin{aligned}
& (\gamma \times d_{iP(i)} + \sum_{j \in \Theta_i} a_j \times d_{ji} + \sum_{k \in D(i)} \gamma \times d_{kP(k)}) - \\
& (a_i \times d_{iP(i)} + \sum_{j \in \Theta_i} a_j \times \min_{l \in (C^{opt} - \{i\})} d_{jl} + \sum_{k \in D(i)} \gamma \times d_{kP'(k)}) \\
= \; & (\gamma - a_i) \times d_{iP(i)} - \sum_{j \in \Theta_i} a_j \times (\min_{l \in (C^{opt} - \{i\})} d_{jl} - d_{ji}) + \\
& \sum_{k \in D(i)} \gamma \times (d_{kP'(k)} - d_{kP(k)}) \\
= \; & (\gamma - a_i) \times d_{iP(i)} - (\sum_{j \in \Theta_i} \psi_j + \sum_{k \in D(i)} \varphi_k) \\
> \; & 0,
\end{aligned}
$$

which contradicts the optimality of the optimal solution. ∎

Payment Mechanism. For each cache node $i \in C^A$, the nodes who benefit from its being cache nodes, i.e., the nodes in $\Theta_i \cup D(i)$, each makes some amount of bid to node i. For cache node $j \in C^{opt}$, it could have multiple ancestor cache node $i \in C^A$, so it needs to make bid to each of them. If this node caches the data, then the bidding node must pay the bided amount to the caching node. The payment mechanism also decides for each caching node in C^A, beyond how much bid it receives that it is willing to cache the data. Using this payment mechanism, we show that both Nash Equilibrium and social optimal can be achieved.

We define the strategy of each node i as $((v^1_i, b^1_i), (v^2_i, b^2_i), ..., (v^{p_i}_i, b^{p_i}_i), t_i)$, where $v^1_i, v^2_i, ..., v^{p_i}_i \in N$, $b^1_i, b^2_i, ...b^{p_i}_i \in R+$, and $t_i \in R+$, indicating i) node i, if a non-cache node, makes b^1_i amount of bid to cache node v^1_i; if a cache node itself, makes bids to each of its p_i ancestor cache nodes, $v^1_i, v^2_i, ..., v^{p_i}_i$, with the amount $b^1_i, b^2_i, ...b^{p_i}_i$ respectively, and ii) node i's threshold value of received bid is t_i beyond which i will cache the data. We use B_i to denote the total amount of bid that node i receives, i.e., $B_i = \sum_{\{j | i = v^x_j\}} b_j$. A node i will cache the data if and only if $B_i \geq t_i$.

The bid of each node is set as follows. For each $j \in \Theta_i$, the amount j bids i is $b_j = \max\{0, \psi_j - nb_i\}$. For each $k \in D(i)$, the amount k bids i is $b_k = \max\{0, \varphi_k - nb_i\}$.

That is, each node bids the amount which it benefits more than the average benefit of the network due to i's caching. For other nodes $l \notin \Theta_i \cup D(i) \cup \{i\}$, the amount l bids i is $b_l = 0$.

The threshold of i t_i is given as:

$$t_i = \begin{cases} 0 & \text{if } i \in C^N \\ \sum_{j \in \Theta_i} b_j + \sum_{k \in D(i)} b_k & \text{if } i \in C^A \end{cases}$$

For all the non-cache nodes in the optimal solution, their threshold is 0 too.

Below we show that above payment mechanism yields social optimal as well as Nash Equilibrium.

Theorem 3 *The payment mechanism reaches Nash Equilibrium, and it yields social optimal for the entire network.*

Proof: We need to show with the payment mechanism, all the nodes in the optimal solution has no incentive to deviate, as long as others stay with their strategies.

First, we show that for node $i \in C^A$, it better off caches the data in spite of the fact that $a_i < \gamma$. We have

$$\begin{aligned}
B_i &= \sum_{j \in \Theta_i} b_j + \sum_{k \in D(i)} b_k \\
&\geq \sum_{j \in \Theta_i} (\psi_j - nb_i) + \sum_{k \in D(i)} (\varphi_k - nb_i) \\
&= \sum_{j \in \Theta_i} \psi_j + \sum_{k \in D(i)} \varphi_k - \Big(\sum_{j \in \Theta_i} + \sum_{k \in D(i)}\Big) \times nb_i \\
&\geq \sum_{j \in \Theta_i} \psi_j + \sum_{k \in D(i)} \varphi_k \\
&\quad -(|\Theta_i| + |D(i)|) \times NB_i / (|\Theta_i| + |D(i)| + 1) \\
&\geq \sum_{j \in \Theta_i} \psi_j + \sum_{k \in D(i)} \varphi_k \\
&\quad -NB_i + NB_i / (|\Theta_i| + |D(i)| + 1) \\
&\geq (\gamma - a_i) \times d_{iP(i)} + nb_i && \text{Definition of } nb_i \\
&\geq (\gamma - a_i) \times d_{iP(i)} && \text{From Lemma 7}
\end{aligned}$$

Above shows that for cache nodes in the optimal solution that are not cache node in the previous Nash Equilibrium, the amount of bids they collect is more than the extra cost they incur due to caching. They better off to caching and has no incentive to deviate.

Chapter 6

Simulation Results

In this chapter we mainly focus on basic caching game, since for payment model, the PoA is always equals to one. We study a 3×4 sensor grid with 12 nodes. One of the 12 nodes is randomly chosen as source node. Access frequency of each node is a random integer number between 1 and 12 inclusive.

Optimal Algorithm. Our optimal algorithm (denoted as OPT) is an exhaustive approach. It works by enumerating all the possible cache node sets in a network of $|V|$ nodes. There are $2^{|V|-1}$ number of possible cache node sets since there is one source node. For each cache node set, finding the minimum spanning tree among its metric closure takes at most $|E| \times \log|V|$. Therefore the time complexity of the optimal algorithm is $O|E| \times \log|V| \times 2^{|V|-1}$.

Figure 6.1 shows the total cost in both NE and OPT, by varying the caching coefficient

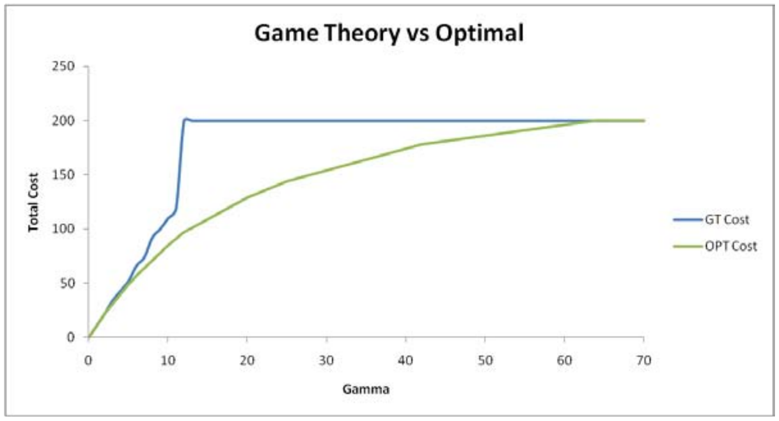

Figure 6.1: Total Cost in basic game, by varying caching coefficient γ.

γ from 0 to 70, and Figure 6.2 shows the corresponding PoA with respect to γ. To more accurately compare NE and OPT, we also list the representative values of both NE and OPT in Table 6.1 in a typical run, including their costs, number of cache nodes, and PoA. It shows that when $\gamma = 0, 1$, the number of cache nodes in NE is 12, while when $\gamma = 2$, the number of cache nodes in NE is 10. This says that there are two nodes with the minimum access frequency of 1 among all the nodes. With the increase of γ, the total costs of both NE and OPT increase, while the number of cache nodes in both NE and OPT decreases. For NE, this is because that a node being a cache node or not only depends on its access frequency and γ, with increase of γ, less and less nodes want to be cache node. However, for OPT, increased γ necessitates less number of cache nodes, even though less cache nodes results in more access cost.

Table 6.1: Representative Values in NE and OPT by Varying γ

γ	NE Cost	# of Cache Nodes in NE	OPT Cost	# of Cache Nodes in OPT	PoA
0	0	12	0	12	1
1	11	12	11	12	1
2	22	10	22	11	1
3	34	9	31	10	1.097
11	119	3	91	7	1.308
12	200	1	97	5	2.062
13	200	1	101	5	1.98
63	200	1	199	2	1.005
64	200	1	200	1	1

When γ increases to 12, there is only one cache node (the source node) in NE, and the total cost is the total access cost in the network, which is the maximum cost of 200 with maximum PoA of 2.062. This says that the maximum access frequency among all the nodes is 11, therefore when $\gamma = 12$, all the nodes access the source node for the data, which costs less compared to caching the data. However, there are still 5 cache nodes in the OPT, to keep the OPT cost low. After that, with the further increase of γ, NE cost stays the same, while the OPT cost still increases. This continues until $\gamma = 64$, when the source node

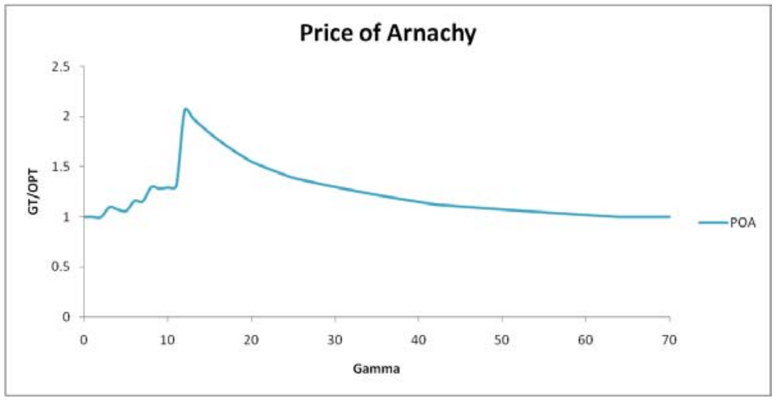

Figure 6.2: Price of anarchy in basic caching game, by varying caching coefficient γ.

becomes the only cache node in OPT, at which time the total costs of both NE and OPT are the total access costs in the network. From then on, PoA stays as 1 with the further increase of γ.

Figure 6.3 shows the change of the number of cache nodes in both NE and OPT, with the increase of γ. It shows that the number of cache nodes in NE is always less than or equal to the number of cache nodes in OPT, indicating the under supply of cache nodes in NE. By studying Figure 6.3 and Figure 6.2 together, we notice when PoA is 1, the number of cache nodes for both NE and OPT are the same. And when PoA gets large, there is a biggest gap between number of cache nodes in game theory and optimal solution. So we can say that optimal solution has a tendency to have more cache nodes than game theory. These extra cache nodes are together bringing down the total cost from game theory.

6.1 Greedy Heuristic

Since *in_Caching* problem is NP-hard, we can not find optimal solution for a large number of sensors. In this section, we design a greedy heuristic strategy to find a good-enough solution in polynomial time, and use it (instead of the brute-force optimal algorithm) to measure total cost, price of anarchy, and number of cache nodes for large network.

Greedy Heuristic. The idea of the greedy heuristic is that we already know that $C^N \subseteq C^{opt}$

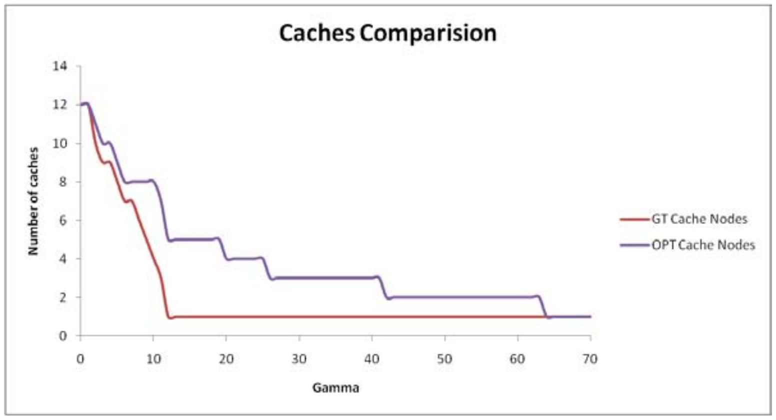

Figure 6.3: Number of cache nodes in basic caching game, by varying caching coefficient γ.

from Lemma 1 (Chapter 4.1). In each iteration we add a node in $(V - C^N)$ to C^N in such a way that the total cost will be reduced the most. We will keep continue adding more nodes until the total cost cannot be reduced further.

The time complexity of the Greedy algorithm is $O(|E| \times \log(|V|) \times |V|^2)$ where $|E|$ is number of edges, $|V|$ is number of vertices in a graph. This can help us to run 300, 400 sensors node in polynomial time efficiently.

With Greedy heuristic, we can experiment with network with more nodes. Below we simulate on a network with 48 nodes. Again, we choose a source node randomly, and access frequency of each node is a random number between 1 and 10 inclusively. Figure 6.4 shows the comparison of total cost between basic game and Greedy. It shows that when γ is small, when γ increases, it quickly reaches the maximum of PoA. It also demonstrates that in a larger scale network, it takes larger caching coefficient γ for the cost of Greedy to be comparable with that of basic game. Figure 6.5 shows the corresponding PoA. Finally, Figure 6.6 shows the number of cache nodes in both cases. Again we observe that the number of cache nodes in basic game is always less than or equal to that of Greedy, indicating the reluctance of nodes to be a cache when its caching coefficient is larger than its access frequency.

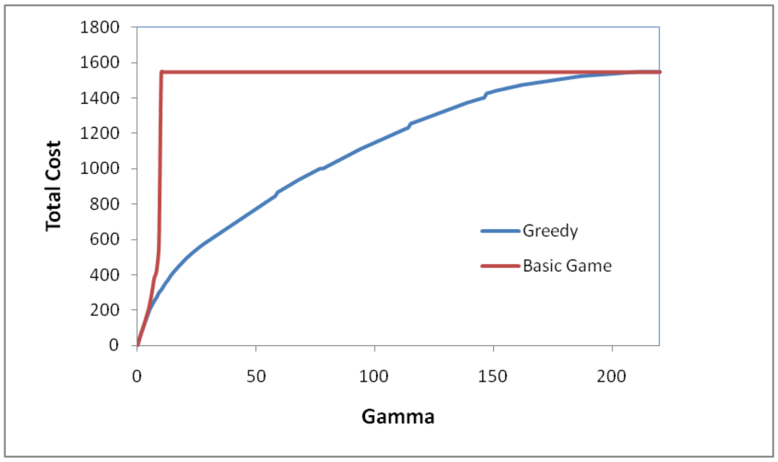

Figure 6.4: Comparison of total costs between basic game and Greedy.

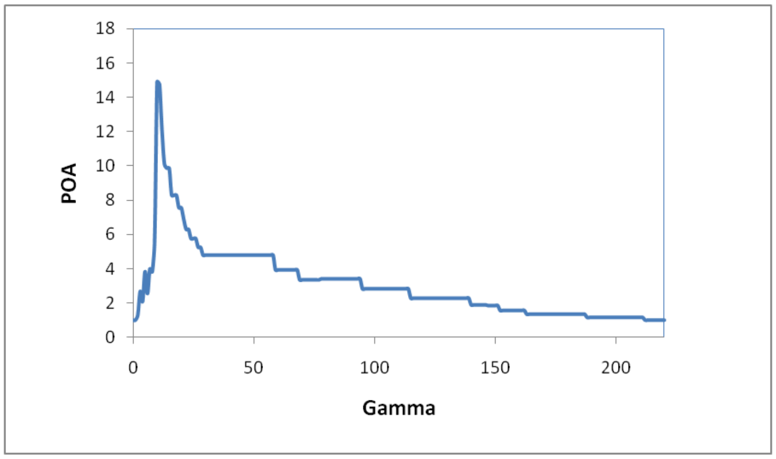

Figure 6.5: Comparison of PoA between basic caching game and Greedy.

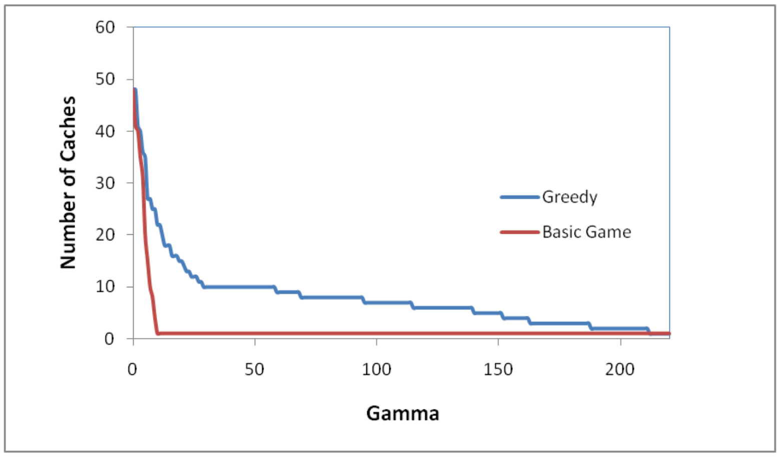

Figure 6.6: Comparison of number of nodes between basic caching game and Greedy.

Chapter 7

Conclusion and Future Work

We apply game-theoretical analysis for the selfish caching in wireless ad hoc networks. Our model considers distance-dependent caching cost, which is different from previous work. We first show a pure Nash Equilibrium exists in our model. We then design a payment model, in which the selfish caching game achieves both optimal cost and Nash Equilibrium simultaneously. Our model is more general and applicable than existing work for such emerging networks as P2P and wireless ad hoc sensor networks. We validate our findings using simulations under various network scenarios. As the ongoing and future work, we are investigating distributed algorithm approach while considering selfish behavior of individual nodes. Our goal is to find efficient mechanism in distributed environment, such that both social optimal and Nash Equilibrium can be achieved.

BIBLIOGRAPHY

Bibliography

[1] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In Proc. of IEEE FOCS 1999.

[2] B.-G. Chun, K.Chaudhuri, H. Wee, M. Barreno, C. Papadimitriou, and J. Kubiatowicz. Selfish caching in distributed systems: A game-theoretic analysis. In Proc. of ACM PODC 2004, pages 21 – 30.

[3] O. Ercetin and L. Tassiulas. Market-based resource allocation for content delivery in the internet. IEEE Transactions on Computers, 52(12):1573–1585, 2003.

[4] Marco Fiore, Francesco Mininni, Claudio Casetti, and Carla-Fabiana Chiasserini. To cache or not to cache. In Proc. of IEEE INFOCOM 2009.

[5] Takahiro Hara and Sanjay K. Madria. Data replication for improving data accessibility in ad hoc networks. IEEE Transactions on Mobile Computing, 5(11):1515–1532, 2006.

[6] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and *k*-median problems using the primal-dual schema and lagrangian relaxation. Journal of the ACM, 48(2):274–296, 2001.

[7] S. U. Khan and I. Ahmad. A pure nash equilibrium based game theoretical method for data replication across multiple servers. IEEE Transactions on Knowledge and Data Engineering, 20(3), 2009.

[8] B.-J. Ko and D. Rubenstein. Distributed self-stablizing placement of replicated resources in emerging networks. IEEE/ACM Transactions on Networking, 13(3):476 – 487, 2005.

[9] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In Proc. of STACS 1999.

[10] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis. Distributed selfish replication. IEEE Transactions on Parallel and Distributed Systems, 17:1401–1413, 2005.

[11] Nikolaos Laoutaris, Georgios Smaragdakis, Azer Bestavros, Ibrahim Matta, and Ioannis Stavrakakis. Distributed selfish caching. IEEE Transactions on Parallel and Distributed Systems, 18:1361–1376, 2007.

[12] Nikolaos Laoutaris, Georgios Smaragdakis, Azer Bestavros, and Ioannis Stavrakakis. Mistreatment in distributed caching groups - causes and implications. In Proc. of IEEE INFOCOM 2006.

[13] P. Mirchandani and R. Francis. Discrete location theory. 1990.

[14] John Nash. Non-cooperative games. Annals of Mathematics, pages 286–295, 1951.

[15] Pavan Nuggehalli, Vikram Srinivasan, and Carla-Fabiana Chiasserini. Energy-efficient

caching strategies in ad hoc wireless networks. In Proc. of MOBIHOC 2003.

[16] C. Papadimitriou. Mistreatment in distributed caching groups causes and implications. In Proc. of ACM STOC 2001.

[17] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. In Proc. of APPROX 2002.

[18] Bin Tang, Samir Das, and Himanshu Gupta. Benefit-based data caching in ad hoc networks. IEEE Transactions on Mobile Computing, 7(3):289–304, 2008.

[19] P. Winter. Steiner problem in networks: A survey. ACM Networks, 17(2):129–167, 1987.

[20] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. IEEE Transactions on Mobile Computing, 5(1):77–89, 2006.

[21] Jing Zhao, Ping Zhang, Guohong Cao, and Chita R. Das. Cooperative caching in wireless p2p networks: Design, implementation, and evaluation. IEEE Transactions on Parallel and Distributed Systems, 99(2):1045–9219, 2009.