COOPERATIVE CONTENT CACHING FOR CAPACITY AND COST MANAGEMENT IN MOBILE ECOSYSTEMS

By

Mahmoud Taghi Zadeh Mehrjardi

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Electrical Engineering

2012

ABSTRACT

COOPERATIVE CONTENT CACHING FOR CAPACITY AND COST MANAGEMENT IN MOBILE ECOSYSTEMS

By

Mahmoud Taghi Zadeh Mehrjardi

The objective of this thesis is to develop an architectural framework of social community based cooperative caching for minimizing electronic content provisioning cost in Mobile Social Wireless Networks (MSWNET). MSWNETs are formed by wireless mobile devices sharing common interests in electronic content, and physically gathering in public settings such as University campuses, work places, malls, and airports. Cooperative caching in such MSWNETs are shown to be able to reduce content provisioning cost which heavily depends on service and pricing dependencies among various stakeholders including content providers, network service providers, and end consumers. This thesis develops practical network, service, and economic pricing models which are then used for creating an optimal cooperative caching strategy based on social community abstraction in wireless networks. The developed framework includes optimal caching algorithms, analytical models, simulation, and prototype experiments for evaluating performance of the proposed strategy. The main contributions are: 1) formulation of economic cost-reward flow models among the MSWNET stakeholders, 2) developing optimal distributed cooperative caching algorithms, 3) characterizing the impacts of network, user and object dynamics, 4) investigating the impacts of user non-cooperation, and finally 5) developing a prototype Social Wireless Network for evaluating the impacts of cooperative caching in a Mobile Social Wireless Networks.

To My Wife, ***Farzaneh Dizaji***
For All Her
Love And Support

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Subir Biswas for his extraordinary support and guidance during my work. He has advised me not only on my research, but also on my career and life. I thus feel fortunate to work with him and proud of being a member of his research group. I would also like to thank my committee Dr. Erik Goodman, Dr. Jian Ren and Dr. Li Xiao for their time and support.

Many thanks goes to Dr. Amir Reza Khakpour and Dr. Alex Liu, for their help and technical feedbacks on designing a distributed firewalling mechanism. I would also like to thank Dr. Eric Torng and Dr. Charles Ofria for their technical feedbacks and their guidance on the problem of finding the upper bound performance of caching.

Thanks must be given to my lab mates Anthony Plummer, Jayanthi Rao, Kris Micinski, Ali Aqel, Muhannad Quwaider, for all of the brainstorming and implementation discussions. Last but not least, I would like give thanks to my beloved wife, Farzaneh Dizaji, and my parents for their unconditional love and support and encouragement throughout my life. It would not have been possible for me to stand where I am now without their help and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 : INTRODUCTION

1.1 Motivation

Recent emergence of mobile devices and wireless-enabled data applications have fostered new content dissemination models in today's mobile ecosystem. A list of such devices includes Apple's iPhone, iPad, Google's Android phones, Amazon's Kindle and electronic book readers from other vendors. The contents related to these devices include phone Apps, electronic books, mp3 music, video clips, news clips etc. The level of proliferation of mobile applications is indicated by the example fact that as of Feb '12, Apple offered over 500,000 Apps that are individually downloadable by smart phone users.

Due to the increase in the number of data-enabled handheld devices and server-based applications, mobile data traffic is growing at an unprecedented rate. According to new studies by some researchers from networking and financial sectors [1–4], by 2016 average broadband mobile data traffic for each user will exceed 2.6 GB per month which is 17-fold increase over the 2011 average of 150Mb per month. The total mobile data traffic in the world is predicted to reach about 10 exabytes per month by 2016 which is 39 times more than the total mobile traffic in 2009. It is not surprising that 70% of this traffic is predicted to be video data.

The increasing demand for more bandwidth raises new issues both in providing ubiquitous Internet access for smart phones and maintaining good voice call quality. In particular, this becomes a critical challenge in the areas without 3G/4G coverage or with

poor quality coverage. In this thesis we propose a pricing based cooperative caching mechanism in order to improve data access for different network topologies, human mobility patterns and communication technologies.

1.2 Current Solutions

There are several solutions to the explosive mobile traffic growth problem. An obvious solution is to scale the network capacity either by building out more cell towers or by upgrading the network to the next generation networks such as Long Term Evolution (LTE) and WiMax. Some providers (e.g. Sprint and Verizon) have already started to support new generation networks such as 4G. However, upgrading the current networks to new technologies requires a huge investment, thus affects the pricing plan. Furthermore, many of the existing hardware and data-enabled mobile devices are not compatible with the new technologies, and therefore upgrading the network infrastructure may not solve the problem for them.

The second solution is using technologies such as *Femtocell* and *Picocell,* which can provide a better coverage and capacity, especially for indoors. A Femtocell is a small cellular base station, typically designed for use in a home or small business. It connects to the service provider's network via a broadband connection (such as DSL or cable) and supports multiple active mobile phones in a residential setting. A Femtocell allows service providers to extend service coverage indoors, especially where access would otherwise be limited or unavailable. This solution is very attractive for mobile operators because in addition to providing good coverage and capacity for their customers it can reduce both capital expenditure and operating expenses. A Picocell has the same functionality of Femtocell with bigger coverage and it is suitable for larger indoor

settings like airports. This technology however has two main limitations. First, it can be only used for small areas, and therefore it is not a permanent large scale solution. Second, although Femtocells provide better coverage, they still need to have a backhaul Internet connection. In other words, Femtocells and Picocells are not able to reduce the Internet bandwidth consumption.

The third solution is to adopt a new usage-based pricing plan which can limit heavy data usage. Switching to a new data plan from a flat rate price has already been started by some providers. For example, Verizon has decided to put a limit on its data plan [5] and MetroPCs offers some prepaid data plan in which user is charged 3 cents per MB. Switching to a more expensive pricing model is not convenient for users and the economic and market competition forces the providers to do a thorough analysis of user behavior and traffic usage before offering a new pricing model. Therefore, switching to new pricing models cannot be done very quickly. Furthermore, even though a new pricing model in short term can reduce the pressure on 3G/4G bandwidth; it is not clear that how it can solve the problem for longer term.

The fourth solution is augmenting (complementing) mobile 3G/4G using WiFi technology. Many multi-interface devices and smart phones give priority to the WiFi interface over the cellular interface in data transmissions [6–10]. This on-the-spot data offloading can reduce the pressure on 3G spectrum by using WiFi connectivity when possible for transferring data. Some providers are also offering incentives to their subscribers to reduce their 3G usage by switching to WiFi at home [11]. This solution relies on the existent wireless hotspots to reduce the bandwidth stress on the 3G network,

but wireless hotspots are not available everywhere and establishing new hotspots requires investment.

1.3 Proposed solution: Pricing Based Cooperative Caching

In this thesis, we introduce a cooperative caching mechanism as a complement to the mechanisms described in previous section. Contrary to those mechanisms, cooperative caching does not require any additional infrastructure and therefore has no additional cost for service providers or end users.

1.3.1 Local Caching

A simple local content caching can always be used within each data-enabled mobile device in order to reduce the 3G/4G bandwidth consumption. In local caching, a new downloaded data item is stored in the local cache in the device to serve the future requests for the same data item. In addition to reducing the bandwidth consumption, local caching can reduce the delay of access and power consumption for mobile devices. *Locality principle* [12] and *Zipf's law* [13] are the two very important and well known facts that support the idea of caching in general.

The locality principle or locality of reference is the phenomenon of the same value or related storage locations being frequently accessed. Temporal locality and spatial locality are two basic forms of reference locality. Temporal locality refers to frequent access to specific data and/or resources within relatively small period of time. Spatial locality refers to the use of data elements within relatively close storage locations. The locality principle has been observed in various fields of Computer Science, e.g. processor

caches, storage hierarchies, Web caches, and search engines. This concept has been also observed for a wide range of online electronic data items [14–16].

According to many studies, the frequency of access request for a set of electronic data items e.g., websites [17–19] and video clips [20–22] follows Zipf's distribution. According to Zipf's law [13], if data items are sorted based on their popularity (i.e. frequency of access request) the access probability of the i^{th} popular item is inversely proportional to its rank. In other words, $P_i \propto \frac{1}{i^{\alpha}}$ where $0 \leq \alpha \leq 1$. As α increases, the access pattern becomes more concentrated on the popular data items. Zipf distribution function belongs to a bigger category of distribution known as power law distributions. When the frequency of an event varies as a power of some attribute of that event (e.g. its size), the frequency is said to follow a power law. This category of distribution is also related to Pareto principle [23], also known as the 80-20 rule. According to the 80-20 rule, for many events, roughly 80% of the effects come from 20% of the causes. The interpretation of this rule for the Zipf distribution implies that 80% of all access is for only 20% of the items. In other words, on an average, by storing only 20% of data items in the cache, 80% of future accesses can be satisfied locally. Therefore, the fact that the request frequency of electronic online content follows Zipf distribution is another strong reason for content caching.

1.3.2 Cooperative Caching

Mobile devices usually have a limited storage capacity and they can store only a few data items. This may affect the performance of caching, especially when data items are relatively large (e.g. video clips and movies). To alleviate this issue, mobile devices

can share the content of their caches together. In cooperative caching, a node not only can access to the content stored in its local cache but it can also search its desired item within stored data items in other caches. The fact that people in the same location tend to share common interests supports the idea of cooperation among their mobile devices. For example, people in a classroom share similar interests on the topic of class or people in a conference have similar interests on the topic of presentation. Under the assumption of having similar interests by users collocated in the same area, it's quite possible for a node to find its desired data item in other nodes' caches.

To facilitate cooperation among mobile devices carried by people who physically gather in public places, a Mobile Social Wireless Network (*MSWNETs*) can be formed using ad hoc wireless connections. For example Figure 1-1 illustrates the location snapshot of a 19-node *MSWNET*. Network partitions can be either multi-hop ad hoc network as shown for partitions 1, 3, and 4, or single hop access point based as shown for partition 2.

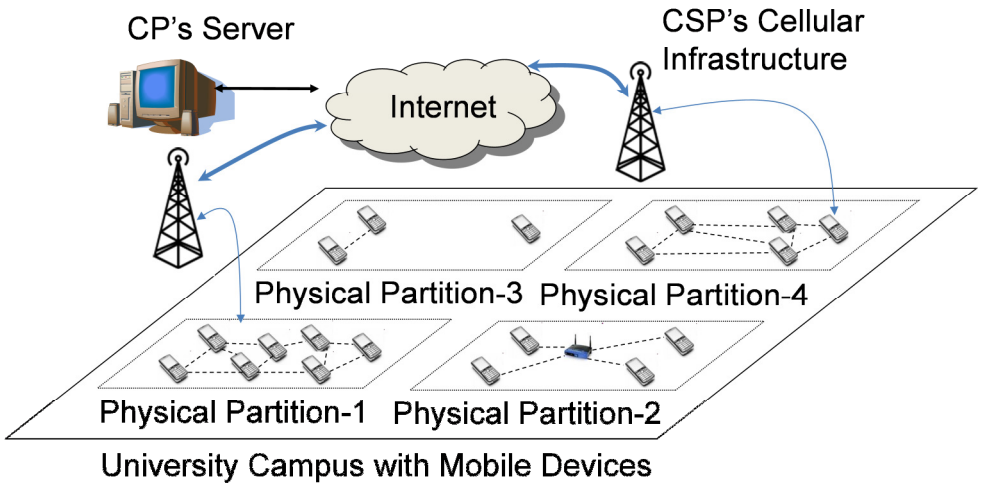


Figure 1-1: Snapshot of an *MSWNET* in a Campus (For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation)

In the conventional download model, a user downloads data items directly from a Content Provider's (CP) server over a Communication Service Provider's (CSP) network. With the local caching a user first searches its local cache before downloading the content from content provider. However, in cooperative caching mechanism, an alternative content access approach would be to search the local *MSWNET* for the requested content after local search for the requested content fails and before downloading it from the CP's server. This new access model can reduce the bandwidth stress on 3G/4G network and improve the content availability when there is no 3G/4G coverage. In this thesis, this mechanism is termed and referred to as *cooperative content caching.*

We believe that the cooperative caching mechanism can be added to the existing solutions (as outlined in Section 1.2) to reduce the 3G/4G bandwidth stress and to improve content availability. The limited storage capacity and sharing of common interests by people collocated in the same place, are the two important facts that support the idea of cooperative caching.

1.4 **Cooperative Caching Components**

In order to deploy a cooperative caching framework each node must support few functionalities. For example, due to limited storage capacity in each mobile device, a replacement mechanism is needed to accommodate a new downloaded data item when a cache is full. Furthermore, every caching node should manage content of its cache based on the other nodes' needs [24–40]. Additionally, a cache resolution mechanism is also needed to find a requested data item among the remote caches. Below, the cooperative caching components are explained in more detail.

1.4.1 Cache Resolution

Cache resolution addresses how to resolve an object request either by finding the object in the local cache or remote caches in the network partition. After a request is originated by a user (i.e. an application on a mobile device), the device first performs a *local search* within its local cache. If it fails to find the requested object, a *network search* is performed for the requested item within the current partition. If this step also fails, and the device has access to the Internet, the requested item will be downloaded directly from the content provider's server.

For searching for an object within the network partition, a flooding-based search mechanism is usually used [28–30]. A Time to Live (TTL) based ring-search can be employed for constraining the scope of resolution. TTL can range from zero to the diameter of the current partition. With zero TTL, a node searches only its local cache for the requested data item. With bigger TTL, more nodes in the partition are searched for the requested data item. TTL also can be large enough to cover all nodes in the partition. A node effectively cooperates with all nodes in its ring search. When a node receives a search request for one of its locally cached objects, it sends a unicast ACK to the requester. Then the requester starts downloading the data item from the responding node.

1.4.2 Cache Management

Cache management refers to policies that control object placement in the caches and determine objects distributed in the partition. Cache management comprises admission and replacement policies.

Admission policy: A node adjusts its cooperative behavior depending on the state of other nodes' caches within its ring search. A simple cooperation policy for a node is not

to store an object if it has already been stored within the partition. Under this policy, the total number of different items stored within the partition can be increased. This in turn can increase the partition item availability for the case no Internet connection is available in the partition.

Cache replacement: To store a new downloaded data item when cache is full a node executes a replacement policy in order to decide as to which data item from its cache should be replaced. Possible replacement policies include Random (RND) [31], Least Recently Used (LRU) [32], and Least Popular Object (LPO) [33]. A complete diagram of a cooperative caching scheme has been provided in Figure 1-2.

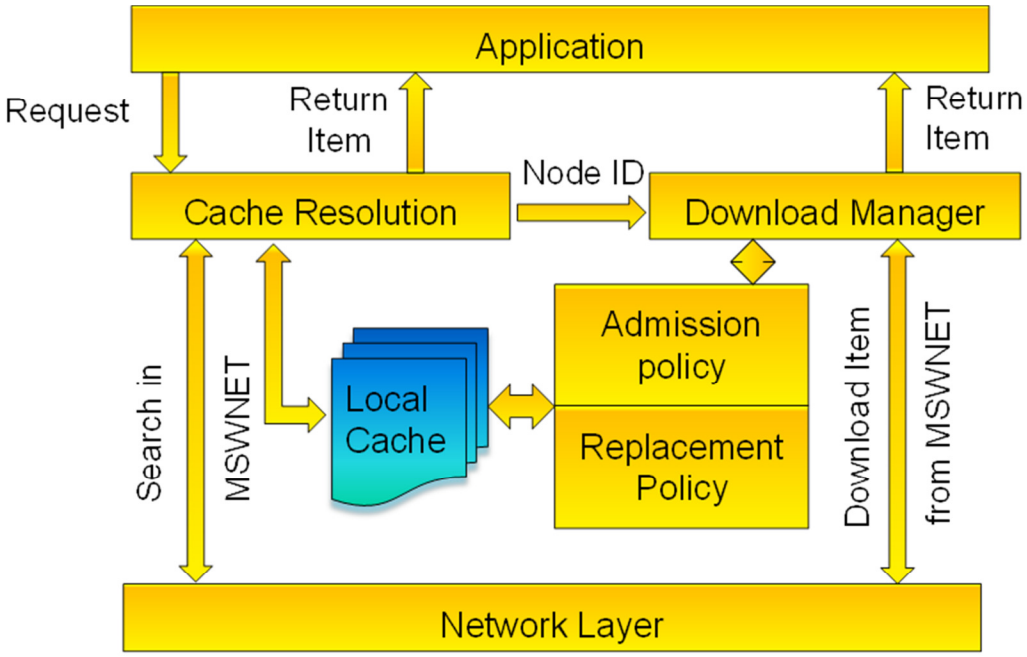


Figure 1-2: Cooperative caching scheme

1.4.3 Cache Consistency

Cache consistency refers to mechanisms that try to keep the content of cached data items and original data items the same. This is a critical component of a caching framework especially when content of data items is subject to frequent changes. There is

a wide range of strategies for maintaining cache consistency. Some of these techniques are specifically designed for mobile ad hoc wireless networks [34–51]. In general, cache consistency strategies fall into two broad categories. In pull-based schemes, a client initiates data validation by polling the content server to see if data has been changed since it was stored. In push-based schemes, the content server initiates the data invalidation by notifying the caching nodes. There are also few hybrid mechanisms in which both strategies are being used. Depending on the type of data, weak or strong data consistencies are enforced. In weak consistency, a stalled data item can be returned to the user, whereas in strong consistency, a stalled data item is never used [52]. Weak consistency works based on a Time-To-Leave (TTL) field which has been associated with all cached data items. A cached data item is returned to a user only if its age in cache is less than TTL. The data item is considered obsolete after its age becomes greater than its TTL.

1.5 **Pricing Model: A Framework for Cooperative Caching**

The main purpose of caching is reducing the bandwidth consumption. Cooperative caching however can also be used to increase the data item availability (when there is no connection to the server). Furthermore, cooperative caching can reduce the access delay for end users or even it can reduce the power consumption of mobile devices. In real life, a different combination of these objectives can also be desired. In this dissertation, we introduce a generalized pricing framework which can model different combinations of these objectives.

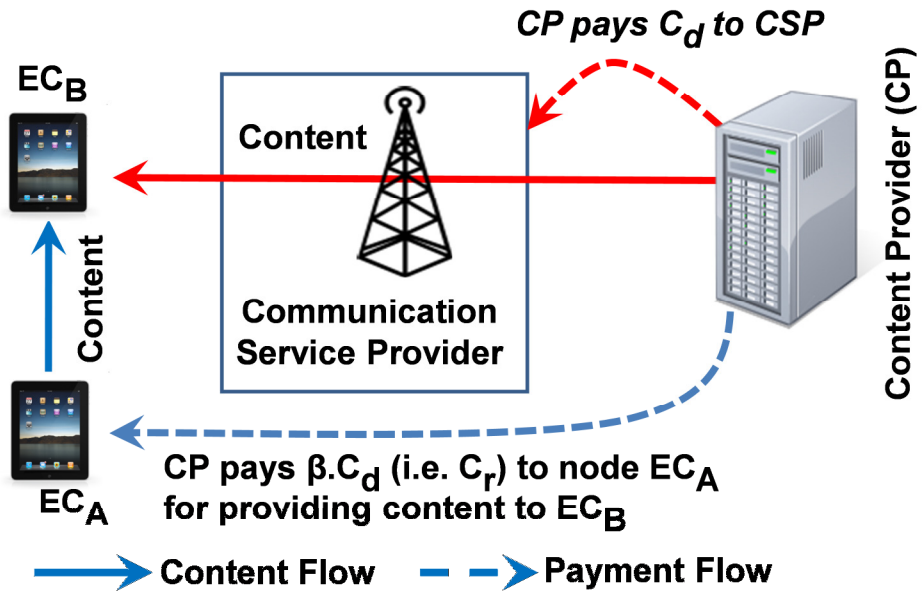


Figure 1-3: Amazon Content and cost flow model

In this thesis, we assume downloading content through a Communication Service Provider's (CSP) network involves a cost which must be paid either by end users or by the Content Provider (CP). Furthermore, we assume that searching and downloading the content from other nodes in the network also require some cost. It is obvious that cooperative caching mechanism can reduce the provisioning cost only when the amount of rebate for a downloaded object is less that the download cost.

The above pricing model has already been used in real life. For example, in the Amazon Kindle electronic book delivery business model, the content provider (i.e. Amazon), pays to Sprint, the communication service provider, for the cost of network usage due to downloaded e-books by Kindle users. Also in order to entice the End-Consumers (ECs) to cache downloaded content and to share it with others, a peer-to-peer rebate mechanism is needed. This rebate can also be distributed among the provider end-

consumer and the end-consumers of all the intermediate mobile devices that take part in object forwarding within the *MSWNET*.

In this thesis we adopt the Amazon Kindle™ pricing model in which the content provider (e.g. Amazon) pays a download cost C_d to the communication service provider (e.g. Sprint) when an end-consumer downloads an object from the content provider's server through the communication service provider cellular network. Also, as shown in Figure 1-3, whenever an end-consumer provides a locally cached object to another user within its *MSWNET*, the provider end-consumer is paid a rebate C_r by the content provider. Optionally, this rebate can also be distributed among the provider end-consumer and the owners of all the intermediate devices that take part in content forwarding. The quantity C_d corresponds to the content provider's object delivering cost when it is delivered through the cellular network, and C_r corresponds to the rebate given out to an end-consumer when the object is found within the *MSWNET*. This framework can be effective for large items for which the network download cost C_d can be also large. Large C_d can ensure large C_r, which makes it practical for an end-consumer to use the received rebate C_r towards its next content purchase from the content provider.

Although in the model used here it is assumed that the network usage cost C_d is paid by the content provider, a very similar problem can be also formulated in a model in which C_d is paid by the end-consumer receiving the content. The rebates, in that case,

will be paid by the content-recipient end-consumer to the content-providing user, and optionally to the intermediate users. This pricing model has been shown in detail in Figure 1-4.

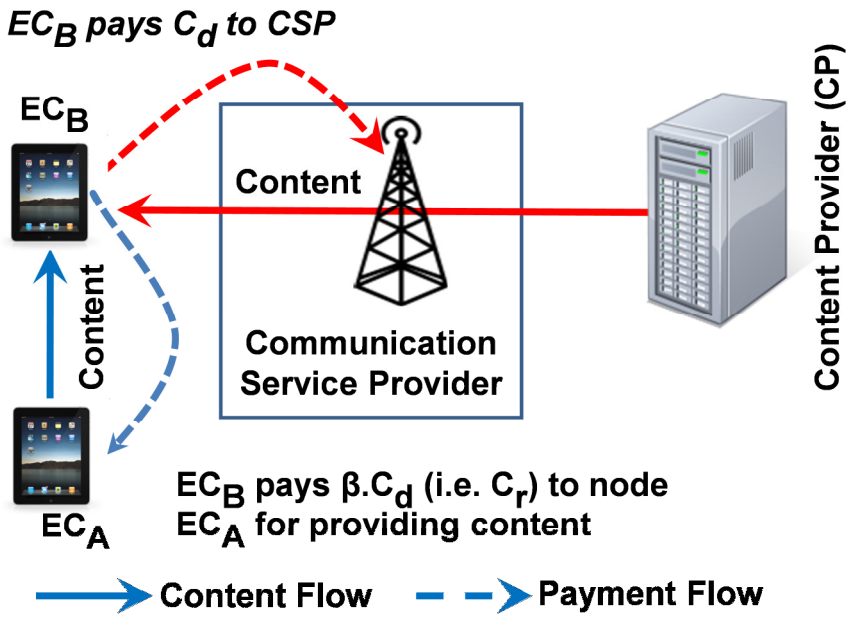


Figure 1-4: Cell phone service provider cost flow model

Note that the cost items, namely, C_d and C_r, do not represent the selling price of an object (e.g. an e-book). The selling price is directly paid to the content provider (e.g. Amazon) by an end-consumer (e.g. a Kindle user) through an out-of-band secure payment system. A digitally signed rebate framework needs to be supported so that the rebate recipient users can electronically validate and redeem the rebate with the content provider. Also, a digital usage right mechanism [53–56] is needed so that an end-consumer which is caching an object (e.g. an e-book) should not necessarily be able to open/read it unless it has explicitly purchased the object from the content provider. We assume the presence of these two mechanisms on which the proposed caching architecture is built. Operationally, the values of C_d and C_r are likely to be set by the

content provider based on its operating cost and revenue models, and the end-consumers are less likely to have any control over those parameters.

The above pricing model is a generalized formulation of the cooperative caching problem. Here we explain how this model can be used for specific purposes.

1.5.1 Minimizing Bandwidth Stress

In order to reduce the 3G/4G bandwidth demand, we must maximize the percentage of requests that can be found within the network partition. In general, finding the best object placement to minimize the bandwidth consumption is not easy. However, in simple cases when the network is stationary and all nodes can communicate with each other (directly or through multi-hop routing) this problem can be formulated in our pricing framework model. We can assume finding an object in the network has no cost while downloading objects from the server requires a certain cost. This means, in the proposed pricing model, we can set $C_r=0$ and then find a solution to minimize the average accessed cost per object. The solution for that problem can be used to minimize the bandwidth stress of a 3G/4G network.

1.5.2 Minimizing Energy Consumption

Although energy consumption is not the primary focus of this thesis, we can model the energy consumption problem using our pricing model. We use the fact that energy consumption of WiFi interfaces and 3G interfaces is different.

Mobile phones usually have multiple network interfaces. A WiFi interface generally provides more bandwidth compared to a 3G/4G interface while they consume almost the same amount of power during data transmission [57]. Therefore, downloading

a data item through a WiFi connection will save energy and increase the battery life because of its smaller transmission time. Furthermore, downloading small contents through 3G network in a periodic manner consumes more energy [58]. The reason is that for a 3G network, a large fraction (nearly 60%) of the energy, referred to as the tail energy, is wasted in high-power states after the completion of a transfer. The measurement in [58] reports that transferring 50KB data over 3G network with a 20-second interval requires 12.5 J energy. This number for a one-hop WiFi connection is around 7.6 J.

In order to minimize the energy consumption for the mobile devices, we can use the proposed pricing model by appropriately setting parameter *Cr*. For example, if we limit the cooperation to 1-hop neighbors, we can set C_r *to 7.6* and C_d to *12.5*. We can then find the best solution for the pricing model problem and apply it to minimize the energy consumption.

1.6 **Handling Human Mobility**

Handheld devices are carried by humans and as people move, the network partitions change dynamically. In other words, the set of nodes in a partition vary over time. Due to the mobility of people, a mobile node may visit a lot of *stranger* (unknown) mobile nodes. Obviously, a mobile node should not change its caching state because of a stranger node that it may never be visited again. In other words, a mobile node should limit its collaboration only to those *familiar* nodes that expected to be visited frequently.

Detecting and limiting the collaboration to the familiar nodes is a crucial part in performance of cooperative caching algorithm when it applied to a human based mobile

network. In this thesis, we study different mobility pattern in order to understand and analyze the difference between the stranger and familiar nodes.

1.6.1 Mobility Patterns

Studying the performance of cooperative caching in social wireless networks requires a real human mobility trace. Due to limited number of available human walk traces and lack of diversity in the existing traces, people use different human mobility generator tools. Statistical analysis of human walk traces in the literature revealed several significant properties. For example, it has been shown that human flights (of walk), pause-time, and inter-contact times follow truncated power-law distribution. Furthermore, it has been observed that individual motilities are confined within some specific areas or point of interests. SLAW [59] and HCMM [60] are the two mobility generator which are able to generate mobility traces that maintain the above statistical properties and therefore we use them in this thesis. In addition to human mobility pattern tools we also use available traces from MIT [61] and UCSD [62] and Heagle [63] projects.

Using these traces we will extract the contact pattern between different nodes and form the group of nodes that belong to the same community. Cooperative caching is then limited only between nodes in the same community.

1.7 Analyzing User Selfishness

According to our proposed pricing model, network usage cost and rebates are paid by the content provider. The scope for earning peer-to-peer rebate may promote selfish behavior in some users. A selfish user is a user that deviates from the network wide

optimal caching policy to earn more rebates. Any deviation from the optimal policy is expected to incur higher network wide provisioning cost.

In this thesis we analyze the impacts of such selfish user behavior on the object provisioning cost and on the earned rebate in a social wireless network (*MSWNET*). In particular, we compare the provisioning cost under presence of selfish nodes with the optimal provisioning cost when all nodes are cooperative.

1.8 Application of proposed Caching

In addition to minimizing the object provisioning cost, the proposed cooperative caching mechanism can be used in other applications. In this thesis, we show how cooperative caching mechanism can be used effectively to reduce the unwanted traffic in MANETs. We also explain how cooperative caching mechanism is able to improve data availability in nodes without Internet connection.

1.8.1 Distributed Firewalling in MANETs

Unwanted traffic wastes significant network bandwidth and the power of resource-constrained wireless nodes [64]. To discard unwanted traffic before reaching destinations, for each wireless node, we need to distribute its firewall rules to other nodes. However, the number of rules that a wireless node can handle is rather limited due to its resource limitations. It is a difficult task for a node to decide which rules should be admitted and enforced given its resource constraints.

The proposed cooperative caching scheme can be used as a solution for discarding unwanted packet in the MANET. A firewall rule is modeled as an object and unwanted

traffic is considered as object requests. The main question then is how to place firewall rules in the network to discard the maximum number of unwanted packets in the network.

1.8.2 Increase Data Availability in MANETs

Due to human mobility and lack of complete coverage by WiFi access points, a *MSWNET* can be susceptible to intermittent disconnections for the Internet. This can result in partitions of devices that can communicate with each other using ad hoc routing protocols, but do not have internet connectivity. This lack of connectivity affects object (content) availability within *MSWNET* partitions for server-based applications such as electronic books and Apps downloads.

The main question is as how to provide high partition level availability while a minimum level of node level availability is guaranteed. High partition level availability ensures that popular objects are available within *MSWNET* partitions when it is disconnected from the Internet, and high node level availability ensures popular objects are available to individual nodes even when they are completely isolated from the rest of the network and Internet gateway. In this thesis we explain how cooperative caching mechanisms can effectively increase the data item availability for both node and partition level.

1.9 Dissertation Objectives

The objective is to design an optimal cooperation policy so that under different network topologies, mobility patterns and rebate to cost ratio, the network wide average per-object provisioning cost is minimized. A key question here is how to store contents in devices such that the network-wide content provisioning cost is minimized. It is very

important to know the set of nodes can potentially form a group (community) to improve the caching performance. In this thesis, we use some well known community detection algorithms that are able to find and group the set of strongly connected nodes.

For contents with varying levels of popularity, a greedy approach for each node would be to store as many distinctly popular contents as its storage allows. This approach amounts to non-cooperation and can give rise to heavy network-wide content duplications. In the other extreme case, which is fully cooperative, a node would try to maximize the total number of unique contents stored within the *MSWNET* by avoiding content duplications. We will show that in a stationary network for a given rebate-to-download-cost ratio, there exists an optimal policy which is somewhere in between those two extremes. This optimal policy can minimize the content provider's cost by striking a balance between greediness and cooperation.

1.10 **Scope of Thesis**

The main goal of this thesis is to provide a cooperative caching strategy that minimizes the object provisioning cost for data-enabled mobile devices such as smart phones, book readers, etc.

To minimize the object provisioning cost we rely on cooperation among mobile devices gathered in a physical setting such as a campus, airport, malls, etc. We propose a cooperative caching framework using which mobile nodes are able to share the content of their caches together. The main intuition behind cooperative caching is the locality principle and common interest between people in the same area. According to the locality principle, people tend to access to the same data that has been used recently and statistically people in the same area are interested in similar content.

Chapter 2 is a survey of other cooperative caching strategies. We review the current cooperative caching strategy used for the wired network and explain why those approaches are not practical in mobile wireless networks. Furthermore, we highlight the differences and advantages of our proposed caching scheme compared to the most recent caching schemes in the literature.

In chapter 3 we propose a cooperative caching strategy to minimize the object provisioning cost in a stationary mobile wireless networks where people in the same area have very similar interests. Furthermore, we assume the request generation rate for all users are the same. Then analytically and experimentally we show that in order to minimize cost per accessed data item in such a network, nodes must limit their level of cooperation.

In chapter 4 we formulate the cost model for a heterogeneous network where users have different request generation rates and different request patterns. An optimized caching strategy will be developed and validated by real human request traces.

Chapter 5 studies the impacts of human mobility on the performance of cooperative caching. Under the varying network partitions, it is very important for a node to collaborate only with nodes in the same community. In this chapter we review some of well-known community detection algorithms in social networks and propose a caching strategy based on the structure of community in the network.

In chapter 6 we analyze the impacts of selfish user behavior on the object provisioning cost and on the earned rebate in a social wireless network (*MSWNET*). In particular, we compare the provisioning cost under presence of selfish nodes with the optimal provisioning cost when all nodes are cooperative.

In chapter 7 we use cooperative caching for increasing the availability of data items in areas without Internet connection.

Chapter 8 introduces another application of the proposed caching strategy which is used to distribute firewalling rules in a social wireless network to stop unwanted packets as early as possible.

Finally, in chapter 9 we summarize the thesis and compile a list of future woks.

Chapter 2 : RELATED WORKS

2.1 Caching in Internet

There are number of studies [12–22] on the characteristics of web proxy traces, which have shown the temporal locality and Pareto principle of the web requests. Relying on these studies web caching has been widely deployed in Internet to reduce the latency observed by web browsers, decrease the aggregate bandwidth consumption of an organization's network, and reduce the load incident on web servers [64-71]. There is a rich body of existing literature on several aspects of cooperative caching including object replacements [31–33], [65–69], cache consistency [52], [70–76], reducing cooperation overhead [24], [77], [78], and cooperation performance in traditional wired networks. Different cooperative caching mechanisms have been introduced for web proxies and file system in [24–27], [77–88]. These cooperation mechanisms can be broadly categorized to hierarchical, directory-based, hash-table and multicast-based approaches. For example, Harvest [25][79] is a hierarchical approach in which a user's request is forwarded to a cache hierarchy till the request is found at some level. In such a hierarchical approach, no information is exchanged between the caches in different level. In a directory-based approach, like Summary Cache [27], each cache exchanges a summary vector of its data items to the other caches. So, during cache resolution a cache forwards the request to a cache that has a copy of the requested data item. Squirrel [96] is a fully decentralized, peer-to-peer cooperative web cache, based on the idea of enabling web browsers on desktop machines to share their local caches by using a hash table. The above schemes have been evaluated and demonstrated significant performance improvement for Web

accessing. However, these schemes are highly dependent on high speed network connections and dedicated cache servers with high computation power and storage. They also require some kind of structure on the network of cooperative nodes. The Mobile Social Wireless Networks (*MSWNET*), which are often formed using mobile ad hoc network protocols, are different in the caching context due to their additional constraints such as topological insatiability and limited resources. As a result, most of the available cooperative caching solutions for traditional static networks are not directly applicable for the *MSWNETs*.

2.2 Cooperative Caching in MANET

There are several studies [28–30], [89–112] on cooperative caching in wireless ad hoc networks [109]. Three caching schemes for MANET have been presented in [96], [101]. In the first scheme, CacheData, a forwarding node checks the passing-by objects and caches the ones deemed useful according to some pre-defined criteria. This way, the subsequent requests for the cached objects can be satisfied by an intermediate node. A problem with this approach is that storing large number of popular objects in large number of intermediate nodes does not scale well. The second approach, CachePath, is different in that the intermediate nodes do not save the objects; instead they only record paths to the closest node where the objects can be found. The idea in CachePath is to reduce latency and overhead of cache resolution by finding the location of objects. This strategy works poorly in a highly mobile environment since most of the recorded paths become obsolete very soon. The last approach in [96], [101] is the HybridCache in which either CacheData or CachePath is used based on the properties of the passing-by objects through an intermediate node. While all three mechanisms offer a reasonable caching

solution, it is shown in [28], [29], [89], [90], [97] that relying only on the nodes in an object's path is not the most efficient approach. Using a limited broadcast based cache resolution can significantly improve the overall hit rate and the effective capacity overhead of cooperative caching.

According to the protocols in [29], [102], [103] the mobile hosts share their cache contents in order to reduce both the number of server requests and the number of access misses. The concept is extended in [30] for tightly-coupled groups with similar mobility and data access patterns. This extended version adopts an intelligent bloom filter based peer cache signature to minimize the number of flooded message during cache resolution. A notable limitation of this approach is that it relies on a centralized mobile support center to discover nodes with common mobility pattern and similar data access patterns. Our work, on the contrary, is fully distributed in which the mobile devices cooperate in a peer-to-peer fashion for minimizing the object access cost.

The authors in [104] propose a cache admission policy in which a node does not cache objects that come from other nodes that are geographically close. It was shown that such distance-constrained admission policies can increase the overall object availability in the presence of network disconnections caused due to node mobility. This protocol attempts to maximize the overall cache hit rate within a partition by exploiting the above distance-constrained cache admission for minimizing the object duplications. [28] proposes a distributed algorithm that exploits highly effective network-wide shared object storage capacity by completely avoiding object duplication across the network nodes (referred to as exclusive caching). In this thesis we have successfully demonstrated that the exclusive caching does not necessarily offer an optimal solution under many cost

formulations. We have proposed an adaptive cache replacement policy that provides the right balance between object duplication and uniqueness to provide the minimum per object provisioning cost under a practical set of network and cost models.

In summary, in most of the existing work, there is a focus on maximizing the cache hit rate of objects, without considering its effects on the overall cost which depends heavily on the content service and pricing models. We formulate different object replacement mechanisms to minimize the provisioning cost, instead of just maximizing the hit rate.

2.2.1 Cache Resolution

Cache resolution schemes reported in [28], [113–115] have proposed various methods to locate objects (i.e. cache resolution) in a mobile network. The protocol in [90] proposes a profile based approach to minimize the cost of search among cooperative mobile terminals trying to access web pages. This and the schemes in [113],[96] propose non-flooding based cache resolution approaches. The main idea behind this kind of mechanisms is to avoid the overhead of flooding based cache resolution. Although improving cache resolution is not the objective of our work, the object replacement approach presented in this work is compatible with all the flooding and non-flooding based resolution mechanisms reviewed above. The overhead of cache resolution is reduced in the broadcast based strategy in [28] by utilizing the concepts of cooperative zones, historical profiles, and hop-by-hop resolution.

2.2.2 Replacement Polices

Cache replacement policies for wireless environments were first studied in the broadcast disk project [116][112]. Authors in [116] propose $\mathcal{PIX}$ replacement policy which considers data access probability and broadcast frequency for object replacement. In [112], replacement decisions are made based on both data access history and retrieval delays. An optimal cache replacement policy, called Min-SAUD, was investigated in [80]. The Min-SAUD policy incorporated various factors that affect cache performance, i.e., access probability, retrieval delay, item size, update frequency, and cache validation delay. Defining benefit as the reduction in total access cost, [105] presents a polynomial-time centralized approximation algorithm that delivers a solution whose benefit is at least 1/4 (1/2 for uniform-size data items) of the optimal benefit.

In [106–108] authors analyze the impact of energy on designing a cache replacement policy. In [106] authors formulate an energy efficient coordinated cache replacement policy, called ECORP, as a 0-1 knapsack problem. It also presents a heuristic algorithm called ECORP-Greedy and an optimal solution called ECORP-OPT to solve the problem. A generalized value function for cache replacement algorithms under a strong consistency model has been introduced in [109]. Their proposed general function can be customized for various performance metrics by making the necessary changes. Authors in [108] addressed the problem of energy-conscious cache placement in wireless ad hoc network. They formulated their problem as an integer linear program and gave solutions to design caching strategies that optimally trade-off between energy consumption and access latency. Another energy-efficient cache replacement strategy is

presented in [117] selects data items with the highest utility to cache in local memory in order to minimize the energy cost at mobile nodes.

A group of location dependent replacement policies are presented in [118–121]. In [121] a detailed location dependent query model and a semantic cache replacement policy FAR (Furthest Away Replacement) are introduced. An evictee is chosen according to the current status of the mobile user. The data which are not in the moving direction and are furthest from the user will be discarded first as they are not needed in the near future. In [43] two cache replacement policies PA (Probability Area) and PAID (Probability Area Inverse Distance) are proposed. Three factors are considered during cache replacement: access probability, data distance and valid scope area. The valid scope of an item value is defined as the region within which the item value is valid. The set of valid scopes for all of the item values of a data item is called the scope distribution of the item. The main idea of the policies is that a promising cache replacement policy should choose its evictee data item with a low access probability, a small valid scope area, and a long distance. In [118], a Mobility Aware Replacement Scheme (MARS), is proposed which considers the temporal scores, spatial scores and the cost of retrieving the data item from the server for object replacement. The temporal score takes into account the most recent update and query time in addition to the query and update rate. The spatial score is based on the scope area and direction of movement. MARS+ [119] is an extended version of this mechanism in which the movement history of mobile nodes or repeated patterns in their paths are taken into account to improve cache replacement accuracy. WPRRP is a Weighted Predicted Region based Cache Replacement Policy [120] in which the data item cost is calculated based on access frequency, valid scope area, data size and

weighted data distance. Authors in [121] propose a replacement policy based on the mobility prediction.

2.2.3 Cache Invalidation Protocols

Due to bandwidth and power constraints in ad hoc networks, it is too expensive to maintain strong cache consistency, and the weak consistency model is more practical and feasible. A simple weak consistency model can be based on the Time-To-Live (TTL) mechanism, in which a node considers a cached item usable if its TTL has not expired, and removes the cached data when TTL expires.

Cache consistency control can be push or pull based. In push based approach, the content server sends invalidation messages to all the cache nodes to indicate the update status of data items whereas in pull based method, the cache node polls the owner to determine whether its cached data item is stale or not.

In [34], three invalidation schemes, namely, TS, AT, and SIG, were presented. Most of the newly proposed invalidation schemes such as [34–51], [122] are variants of these basic schemes. They differ from one another mainly in the organization of contents and the mechanism of uplink checking. All of these invalidation schemes incur certain cache validation delay for ensuring data consistency before the data is used.

In location-dependent information services, there is yet another kind of cache invalidation, where a previously cached data instance may become invalid when the client moves to a new location. In [50], three schemes for this kind of location-dependent cache invalidation are proposed.

2.3 Analysis of Selfishness

The authors in [123–125] investigate the impacts of selfishness and mistreatment on caching. A mistreated node is a cooperative node that experiences an increase in its access cost due to the selfish behavior by other nodes in the network. In [126] authors study selfishness in a distributed content replication strategy in which each user tries to minimize its individual access cost by replicating a subset of objects locally (up to the storage capacity), and accessing the rest from the nearest possible location. Using a game theoretic formulation, the authors prove the existence of a pure Nash equilibria under which network reaches a stable situation. Similar approach has been used in [127] where authors modeled distributed caching as a market sharing game. Our work in this thesis has certain similarity with the above works as we also use a monetary cost and rebate for content dissemination in the network. However, as opposed to using game theoretic approaches, we propose and prove an optimal caching policy. Additionally, the pricing model of our work which is based on the practical Amazon Kindle business model is substantially different and more practical compared to those used in [126], [127].

2.4 Data replication

Data replication is a very similar mechanism to data caching which can be used In MANETS to reduce the average latency and to save wireless bandwidth in a mobile environment. Hara [128] proposed a replica allocation method to increase data accessibility in MANETS. According to [128] a mobile node maintains a limited number of popular data items. Replicated date items are relocated periodically based on access frequency of each mobile node and access frequency of its neighbors. Consistency of replicated data is further considered in [129-131]. An extended version of this replication

mechanism has been presented in [129] by considering a periodic updates and integrating user profiles consisting of mobile users' schedules, access behavior, and read/write patterns.

Replicating popular data can increase the data accessibility since a mobile node cannot access data when it is isolated from the others [129], [130]. However, due to the limited size of data can be stored in a mobile node simply replicating data items cannot fulfill users requirements. To overcome the limited information availability in MANET, authors in [134],[90] proposed a cooperative caching scheme to increase data accessibility by peer-to-peer communication among mobile nodes, when they don't have direct access to the server. The protocol in [90] is implemented on top of Zone Routing Protocol (ZRP). [135],[136] suggested the 7DS architecture to share and disseminate information among users. It operates either on a prefetch mode, based on the information and user's future needs or on an on-demand mode, which searches for data items in a single-hop multicast basis. Unlike other cooperative caching mechanisms this strategy focuses on data dissemination, and thus, the cache management including a cache admission control and replacement policy is not well explored.

2.5 Firewall Rule Caching

We utilize our cooperative caching scheme to deploy a distributed firewalling mechanism in MANETs. There are few other works that address the firewalling in MANETs but none of them uses the rule caching.

Maccari *et. al.* proposed a firewalling scheme for mesh networks in [131]. In this scheme, all accepted packets for each node are represented by a bloom filter. They use a bloom filter to send the list of accepted packets to all nodes in the network. Thus, when a

node wants to forward a packet, it queries the packet from all bloom filters it has received from other nodes. If it is found, the packet is forwarded; otherwise, it is discarded. Due to large number of accepted packets, the bloom filter is very big (it could be in order of GB or TB). To deal with this problem, they only consider packets with class C IP addresses and port numbers less than 1024, which is a considerable limitation of their work. The authors extend their work in neira08mesh to support stateful firewalls and they use d-left counting bloom filter with handover support.

Alicherry *et. al.* presented a traffic authentication framework for MANETs in [132], [133]. In this framework, a set of trusted nodes is appointed as group controller that is responsible for distributing token policies. When a node wants to access an authorized service on a destination, the source node sends the corresponding token policy to the destination. The destination notifies the source node as well as all intermediate nodes along the path to the source about the amount of allocated bandwidth for the requested session.

Chapter 3 : OPTIMAL CACHING FOR HOMOGENOUS NETWORKS

3.1 Motivation

In conventional data access model, a data enabled mobile device (e.g. smart phone, Amazon kindle) downloads an electronic data item from the server through a communication service provider network (e.g. 3G/4G network). The communication cost of this download must be paid by end user or content provider. In this chapter we adopt the Amazon Kindle^{TM} pricing model in which the Content Provider (e.g. Amazon the CP) pays the download cost of an electronic item to the Communication Service Provider (CSP). The object provisioning cost can be reduced when mobile devices collaboratively share their content to each other. To encourage the end-consumers to participate in this cooperative caching, a content provider pays a rebate to an end-consumer when it provides a data item to another device.

A key question for cooperative caching is how to store contents in nodes such that the average content provisioning cost in the network is minimized. In this chapter we show that in a stationary mobile wireless network an optimal caching strategy exists which can minimize the average cost per accessed object.

3.2 Network Model

To facilitate cooperation among mobile devices carried by people who physically gather in public places, a Mobile Social Wireless Network (*MSWNETs*) can be formed using ad hoc wireless connections. For example Figure 3-1 illustrates the location

snapshot of a two physical partitions in a 6-node *MSWNET* setting.

In this chapter we consider two types of *MSWNET*s. The first type involves stationary [134] *MSWNET* partitions. This means that after a partition is formed, it is maintained for sufficiently long so that the cooperative object caches can be formed and reaches steady states. After the optimality and performance of the proposed caching mechanism is evaluated for this stationary case, we investigate the second type to explore as to what happens when the stationary assumption is relaxed. To investigate this effect, caching is applied to *MSWNETs* formed using human interaction traces obtained [135] from a set of real *MSWNET* settings.

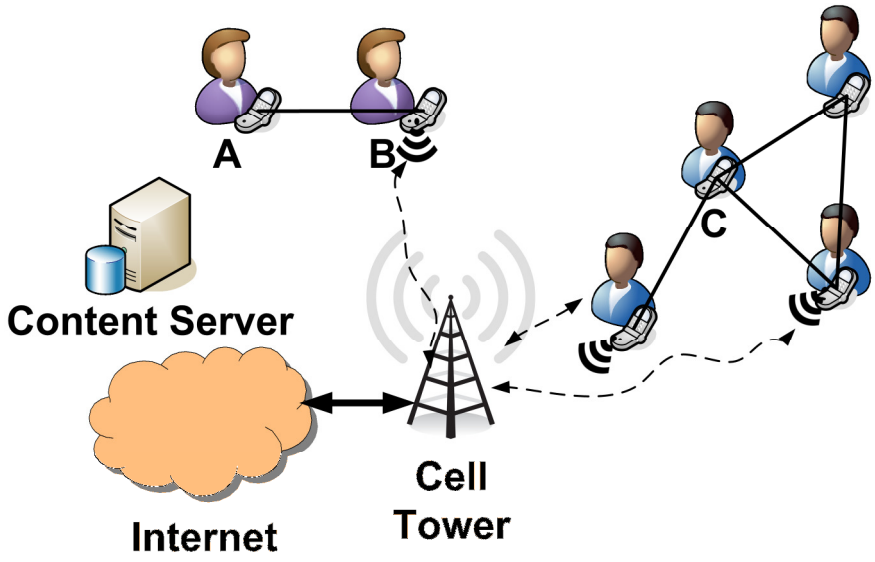


Figure 3-1: Snapshot of two physical partitions in a social wireless network

3.3 Pricing Model

We use a pricing model similar to the Amazon Kindle e-book delivery in which the content provider (e.g. Amazon) pays a download cost C_d to the communication service provider (e.g. Sprint) when an End-Consumer (EC) downloads an e-book from the CP's server through the CSP's cellular network. Also, whenever an EC provides a cached

object to another EC within its *MSWNET* partition, the provider EC is paid a rebate C_r by the CP. Alternatively, this rebate can be distributed among the provider EC and the ECs of all the intermediate mobile devices that take part in object forwarding. The flow of content and cost has been shown in Figure 3-2.

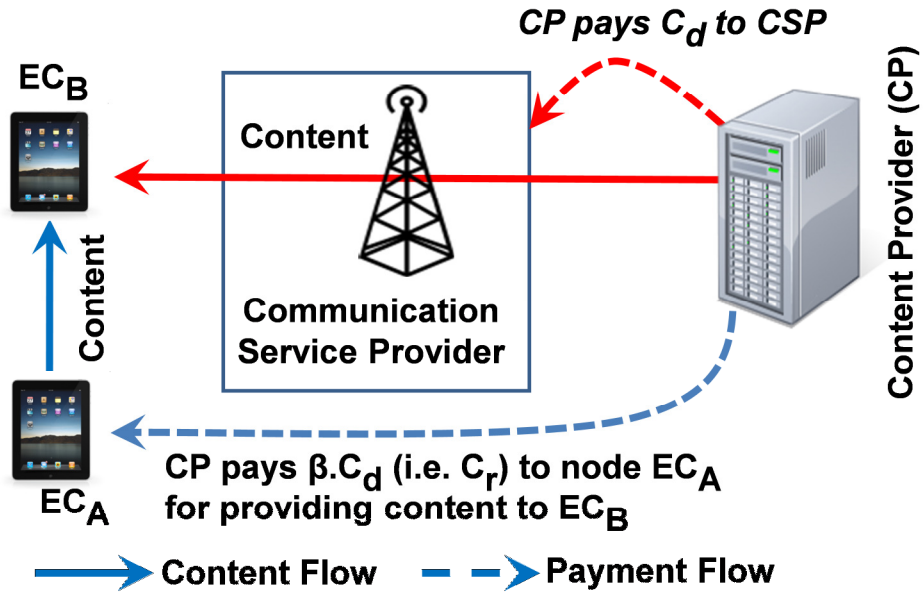


Figure 3-2: Amazon Content and cost flow model

3.4 Request Model

A server-tagged object popularity [136], [137] and a Zipf distribution [19] based object request model, which is widely used in the literature for modeling popularity based online object request distributions, have been used. We also assume all mobile nodes generate the same number of requests per unit time and all users have the same interest. This means the probability of generating a request for a given item is the same for all nodes.

3.5 Search Model

After a request generated within a mobile device, it first search its local cache for the requested item. If local search fails, a broadcast request is sent over the MSWNET to search the item in the other nodes in the partition. If the network search for the data item also fails the requested item will be downloaded from content provider server through communication service provider network. (e.g. 3G network).

3.6 Cost of Content Provisioning

Table 3-1 shows the list of notations used in the subsequent derivations.

Table 3-1: Notations used in cost computation

m	Number of ECs in an *MSWNET* partition
C	Cache size (No. of objects can be stored locally)
C_d	Cost paid by CP (or End consumer) to CSP per object download
C_r	Rebate paid by CP (or End consumer) to publisher node per object
$\beta=C_r/C_d$	Rebate to download cost ratio
α	Zipf parameter ($0 < \alpha < 1$)
N	Total number of objects in the network
P_L	*Local* hit rate; probability that a requested object is found in the local cache
P_V	Remote hit rate; probability that a requested object is found in the partition
P_M	Miss rate; probability that a requested object is not found in the partition
p_i	Popularity of object ‘i’ in the network
S_j	Set of objects stored at node ‘j'
n_i	Number of copies of object ‘i’ in the network
T	Least popular object stored in the network

Let P_L be the probability of finding a requested object in the local cache (i.e. *local* hit rate), P_V be the probability that a requested object can be found in the network (i.e. *Remote* hit rate) after its local search failed, and P_M be the probability that a requested object is not found in the network. We can write P_M in terms of P_V and P_L as:

$$P_M = 1 - P_L - P_V \tag{3-1}$$

The provisioning cost for an object is zero if the object is found in the local cache, C_r when it is found in the *MSWNET*, and C_d when it is downloaded from the CP's server through the CSP's network. Thus, average provisioning cost is:

$$Cost = P_V C_r + P_M C_d \tag{3-2}$$

Expressing C_r/C_d as β and substituting P_M from Eqn. 3-1, cost can be expressed as:

$$Cost = (1 - (1 - \beta)P_V - P_L)C_d \tag{3-3}$$

Let m be the number of devices within an *MSWNET* partition, and S_j be the set of objects stored in device-*j* $(1 \leq j \leq m)$. With p_i $(1 \leq i \leq N)$ as defined in Eqn. 1-1, the probability of finding an object in device-*j*'s cache can be written as $P_L^j = \sum_{i \in S_j} p_i$. The resulting probability of finding the object at any given device in the

partition is $\sum_{j=1}^{m} P_L^j / m$ or $\sum_{j=1}^{m} \sum_{i \in S_j} p_i / m$ (Recall that the request rate of all nodes is the same). This is the average local hit rate P_L, and can be simplified as:

$$P_L = \frac{1}{m} \sum_{i=1}^{N} n_i p_i \tag{3-4}$$

where n_i represents the number of copies of object-*i* within the partition. If *C* is the available cache size (i.e. the number of objects that can be stored) at each mobile device, then the maximum number of objects that can be stored within a *MSWNET* partition is mC.

With any popularity based object request model (e.g. Zipf), a meaningful cooperative caching approach must ensure the following constraint at steady state.

Popularity storage constraint: An object should not be cached in a partition when at least one object of higher popularity is missing in the partition. Meaning, object *i* cannot be cached while object *k* (*k* < *i*) is missing. With this constraint:

$$P_L = \frac{1}{m} \sum_{i=1}^{mC} n_i p_i \tag{3-5}$$

Note that the *popularity storage constraint* does not dictate the level of allowed object duplication in a partition. It is allowed to have multiple copies of the same object within the partition. Object duplication within a node, however, is not beneficial and not allowed.

Let $\mathcal{S}$ represent the set of all stored objects in a partition. The probability of finding an object in the partition can be expressed as $\sum_{i \in \mathcal{S}} p_i$, or $\sum_{i=1}^{T} p_i$, where the *T-th* popular object is the least popular one stored in the partition. The quantity $\sum_{i=1}^{T} p_i$

represents the overall cache hit rate in the partition which is equal to $1 - P_M$. Substituting $\sum_{i=1}^{T} p_i$ for $1 - P_M$ and the value of P_L from Eqn. 3-5 in Eqn. 3-1, we can write

$$P_V = \sum_{i=1}^{T} p_i - \frac{1}{m}\sum_{i=1}^{mC} n_i p_i$$

Using Eqn. 3-3, the cost expression can be written as $\left(1 - (1 - \beta)\left(\sum_{i=1}^{T} p_i - \frac{1}{m}\sum_{i=1}^{mC} n_i p_i\right) - \frac{1}{m}\sum_{i=1}^{mC} n_i p_i\right) C_d$ which can be further simplified as:

$$Cost = \left(1 - (1 - \beta)\sum_{i=1}^{T} p_i - \beta\frac{1}{m}\sum_{i=1}^{mC} n_i p_i\right) C_d \qquad (3\text{-}6)$$

3.7 Minimizing Object Provisioning Cost

For a given β, the cost in Eqn. 3-6 is a function of the vector $\vec{n} = < n_1, n_2, \ldots, n_N >$ where n_i shows the number of copies of object '*i*' in the *SWNET* partition in question. An object placement $\vec{n}$ is optimal when it leads to minimum object provisioning cost in Eqn. 3-6. In this section, we aim to determine the optimal $\vec{n}$.

Lemma 1: With any popularity based object request model (e.g. Zipf), the optimal placement approach must ensure the following constraint at steady state:

An object should not be stored in a partition when at least one object of higher popularity is missing in that partition. That is, object *i* (i.e. *i-th* popular object) cannot be cached while a higher popularity object *k* (*k* < *i*) is missing. This is referred to as *popularity storage constraint*.

Proof: Let us assume that there is an optimal placement which minimizes the object provisioning cost in Eqn. 3-6 and violates the *popularity storage constraint*. It means there is a missing object '*i*' in the *SWNET* (i.e. $n_i = 0$) while a less popular object '*j*' is present (i.e. $j > i, n_j > 0$).

Using Eqn. 3-6, it can be shown that if a less popular object '*j*' is replaced with the missing object '*i*', the cost will be lower. This contradicts the assumption and therefore, the optimal object placement must preserve the *popularity storage constraint*.

Now let us assume that '*T*' is the least popular object in the optimal solution. According to the popularity storage constraint, there is at least one copy of objects '*1*' to '*T*' in the partition. Therefore, Eqn. 3-6 can be written as:

$$Cost = \left(1 - (1-\beta)\sum_{i=1}^{T} p_i - \beta \frac{1}{m}\sum_{i=1}^{T} n_i p_i\right) C_d \qquad (3\text{-}7)$$

Lemma 2: In the optimal object placement, an object *k* (i.e. *k-th* popular object) should not be duplicated unless all other objects with higher popularity have been duplicated in all nodes.

Proof: According to the *storage popularity constraint* in the optimal solution, at least one copy of object '*1*' to object '*T*' exists. Since object '*T*' is the least popular object in the optimal solution, Eqn. 3-7 can be rewritten as:

$$Cost = 1 - \sum_{i=1}^{T}\left(1 - \beta - \frac{\beta}{m}\right)p_i - \frac{\beta}{m}\sum_{i=1}^{T}(n_i - 1)p_i$$

Now, let $n_\ell \neq n_k, 1 < n_\ell, n_k < m,$ and $\ell < k$. It can be observed that by increasing n_ℓ and by reducing n_k it is possible to lower the cost. This can lead to the following claim: while there is room for increasing the number of copies of object ℓ (i.e.

$n_\ell < m$), less popular objects (e.g. object k, $k > l$) should not be duplicated. Following the above logic, we cannot duplicate object '*2*' unless we have duplicated object '*1*' in all nodes (i.e. $n_1 = m$). Similarly, we cannot duplicate objet 'i' unless we have already duplicated more popular objects in all nodes.

Claim: The optimal object placement $\vec{n}$ has the following properties:

1) $n_i = m \; for \; 1 \leq i \leq \ell$, where ℓ is the least popular duplicated object in the network, and its value should be determined based on β. One copy of objects $1 \ldots \ell$ will be stored in all nodes.
2) $n_i = 1 \; for \; \ell + 1 \leq i \leq T$, where $T = NC - N\ell + l + 1$. This means the remaining space of caches is filled with unique objects.
3) $n_i = 0 \; for \; i > T$

Proof: According to *Lemma 1*, There must be at least one copy of objects $1 \ldots T$ in the network (i.e. there is no missing object). *Lemma 2* states that an object should not duplicated before all other objects with higher popularity have been duplicated in all nodes. This means if ℓ is the least duplicated popular object in the network, there should be m number of copies of objects $1..\ell$ in the network.

Note that the above analysis does not help deciding the value of ℓ, or the set of objects that need to be duplicated for the optimal object placement solution. It only shows that if the optimal solution requires duplication, it must be across all nodes. In the next section we show how to determine the value of ℓ.

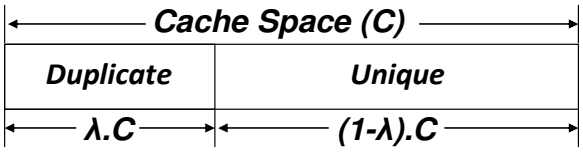


Figure 3-3: Cache partitioning in Split Cache policy

3.8 **Cooperative Split Cache Mechanism**

To realize the optimal object placement as described in Section 3.7 we propose the following *Split Cache* policy in which the available cache space in each device is divided into a duplicate segment (λ fraction) and a unique segment (see Figure 3-3). In the first segment, nodes can store the most popular objects without worrying about the object duplication and in the second segment only unique objects are allowed to be stored. Parameter λ in Figure 3-3 ($0 \leq \lambda \leq 1$) indicates the fraction of cache that is used for storing duplicated objects.

With the *Split Cache* replacement policy, soon after an object is downloaded from the CP's server, it is categorized as a *unique* object as there is only one copy of this object in the network. Also, when a node downloads an object from another node in the network, that object is categorized as a *duplicated* object as there are at least two copies of that object in the network. Because of limited cache storage, a node cannot store all downloaded objects. Therefore, to store a new incoming object a node has to select and evict another object from its cache. For storing a new *unique* object, the least popular object in the whole cache is selected as a candidate and it is replaced with the new object if it is less popular than the new incoming object. For a *duplicated* object, however, the *evictee* candidate is selected only from the first duplicate segment of the cache. In other words, a unique object is never evicted in order to accommodate a duplicated object. The *Split Cache* object replacement mechanism realizes the optimal strategy established in Section 3.7. With this mechanism, at steady state all devices' caches maintain the same object set in their duplicate areas, but distinct objects in their unique areas. The pseudo code of *Split Cache* replacement policy is shown in Algoritm-1.

INPUT: Object O_{new}
IF (*O_{new} is downloaded from another node*)
$O_{min} = the\ least\ popular\ obj\ in\ the\ \boldsymbol{duplicate}\ area$
ELSE
$O_{min} = The\ least\ popualar\ obj\ in\ the\ \boldsymbol{entire}\ cache$
END
IF ($O_{new}.popularity > O_{min}.popularity$)
$\boldsymbol{replace}\ O_{min}\ with\ O_{new}$
END

Algorithm 1: Split Cache object replacement policy

Cost for Cooperative Split Cache: To compute the provisioning cost for *Split Cache* we first calculate P_L (local hit rate) and P_V (remote hit rate) and then substitute their calculated values in Eqn. 3-3. To facilitate calculation of P_L and P_V we define a function f(k) which represents the probability of finding an arbitrary object within a device's cache that is filled with the k most popular objects. This function can be expressed as $\sum_{i=1}^{k} p_i$. Substituting p_i for the Zipf distribution, $f(k) = \sum_{i=1}^{k} p_i \approx \int_1^k \frac{\Omega}{i^\alpha} di =$ $\Omega \frac{k^{(1-\alpha)}-1}{1-\alpha}$. Now, considering $\Omega = 1/\sum_{i=1}^{N} p_i \approx 1/\int_1^k \frac{1}{i^\alpha} di = \frac{1-\alpha}{N^{(1-\alpha)}-1}$, we can write:

$$f(k) = \frac{k^{(1-\alpha)}-1}{N^{(1-\alpha)}-1} \tag{3-8}$$

Local Hit Rate P_L: At steady state, total number of unique objects stored in the partition is equal to $mC(1-\lambda)$. Also, the number of duplicated objects is equal to λC.

Therefore, the total number of different objects stored in the partition is $\lambda C + m C(1 - \lambda)$. Probability that a device can find a new requested object in its local cache is equal to:

$$P_L = H_D + \frac{H_U}{m} \tag{3-9}$$

where $H_D = f(\lambda C)$ corresponds to the cache hits contributed by the objects stored in the duplicate area of cache and $H_U = f\left[\left(\lambda + m(1 - \lambda)\right)C\right] - f(\lambda C)$ represents the hit rate contributed by the unique objects (in the partition) which are assumed to be uniformly distributed over all m devices' caches.

Remote Hit Rate P_V: It is equal to the hit probability contributed by the objects stored in the unique area of all devices in the partition, minus the unique area of the local cache. This can be expressed as:

$$P_V = \frac{m-1}{m} H_U \tag{3-10}$$

Substituting the value of P_L and P_V from Eqns. 3-8 $P_L = H_D + \frac{H_U}{m}$ (3-9 and 3-10 in Eqn. 3-3, the cost can be simplified as:

$$Cost = \left(1 - \left(\frac{(1-\beta)m+\beta}{m}\right) H_U - H_D\right) C_d \tag{3-11}$$

Using Eqn. 3-8 to expand H_U and H_D, Eqn. 3-11 can be written as a function of λ. By equating the derivative of the cost expression to zero, we can compute λ_{opt} at which cost is minimized. The values and trend of λ_{opt} for different system parameters including C, m, α and β are presented in Section 3.10.

3.9 Handling Objects with Different Size

For the sake of modeling simplicity, so far we have assumed that all objects have the same size. In this section, the minimum-cost object replacement mechanism is extended for scenarios in which objects can have different sizes. In such situations, in order to insert a new downloaded object '*i*' from the CP's server, instead of finding the least popular object, a node needs to identify a set of objects ψ in the cache. The set ψ should be identified such that the quantity $\sum_{j\in\Psi} p_j$ is minimized while $\sum_{j\in\Psi} p_j < p_i$ and $\sum_{j\in\Psi} x_j > x_i \sum_{j\in\Psi} x_j > x_i$; the quantity x_i shows the size of object '*i*'. This is a traditional knapsack problem for which a number of heuristics based solutions are available in the literature. If a set ψ, satisfying the above conditions, is found, then all objects in that set are evicted from the cache to accommodate the new incoming object; otherwise the incoming object '*i*' is not admitted. When an object is downloaded from another node in the *MSWNET*, the members of ψ can be selected only from the objects stored in the duplicate area of the cache. Note that dimensioning of the split factor λ with varying object size is not addressed in this chapter.

3.10 Evaluation in Static Partitions

The performance of *Split Cache* in static partitions was evaluated using the analytical expressions in Section 3.8, and then via ns2 network simulation. For simulation, a flooding based object search mechanism has been implemented using the baseline AODV [138] route discovery syntaxes. Baseline experimental parameters are summarized in Table 3-2.

Table 3-2: Baseline simulation parameters

Number of ECs in a static partition(V)	40
Download cost (C_d)	10
Rebate-to-download-cost ratio (β)	$0 \le \beta \le 1$
Cache size in each mobile device (C)	50
Zipf parameter (α)	0.8
Object population (N)	5000
Warm up phase to reach steady state	2000 requests
Total simulation duration	10000 requests

3.10.1 Hit Rates and Provisioning Cost

Figure 3-4(a) depicts the impacts of λ on the hit rates. Smaller λ values lead to fewer copies of the popular objects within the local cache and the subsequent low local hit rates. Larger number of unique objects in the partition, however, leads to higher remote hit rates for smaller λ values. Since with larger λ, more popular objects are duplicated, the likelihood of finding objects locally improves, leading to higher P_L values.

The miss rate P_M depends on the total number of unique objects in the *MSWNET* partition, which increases with higher duplications when λ is increased. The excellent agreement between the analytical and the simulation results in Figure 3-4(a) validates the analysis in Section 3.8.

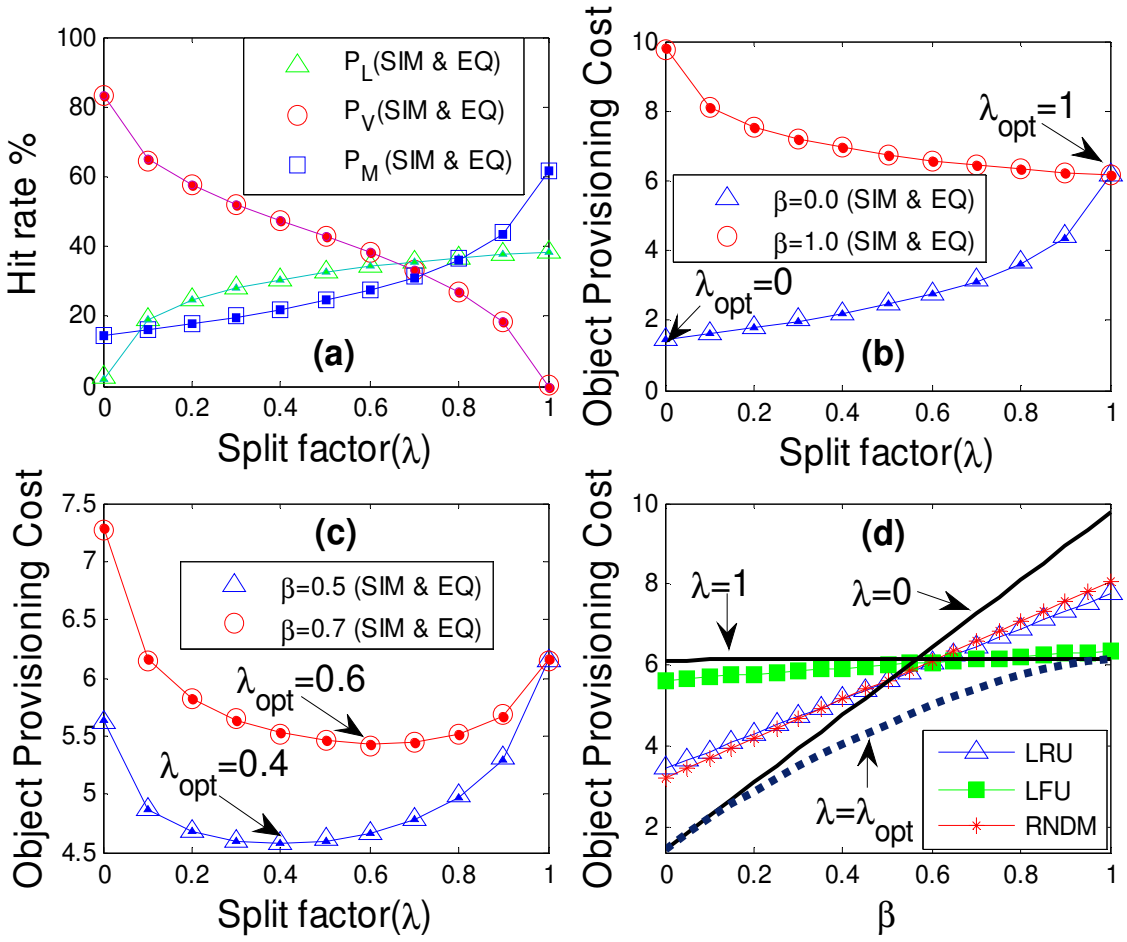


Figure 3-4: Hit rates and cost for *Split Cache* and traditional policies

Figure 3-4(b and c) depict the provisioning cost as a function of λ. When $\beta = 0$ (i.e. per object credit C_r=0), the cost expression in Eqn. 3-3 reduces to $Cost = P_M C_d$. Meaning, for a given C_d, the cost depends only on P_M which reduces as λ reduces. Therefore, when $\beta = 0$, λ=0 results in minimum P_M and consequently the minimum cost. When $= 1$, meaning the r ebate is same as the download cost C_d, the expression in Eqn. 3-3 reduces to $Cost = (1 - P_L)C_d$, indicating that it depends only on the local hit rate for a given C_d. This explains as to why the cost decreases with increasing λ, whereas

the local hit rate increases. Intuitively, when $C_r = C_d$, there is no advantage of fetching objects from the *MSWNET*. The only way to reduce cost in this situation is to maximize P_L. Observe in Figure 3-4(c) that for both $\beta = 0.5$ and $\beta = 0.7$, the cost reduces initially for increasing λ but after a critical $\lambda = \lambda_{opt}$, the cost starts to increase. This critical point can be found numerically from Eqn. 3-11. It was established in Section 3.7 that starting from the state of zero partition-wide duplication, if the iterative duplication/replacement process stops at the correct point, the cost can be minimized. This translates to finding the appropriate level of duplication, which is decided by λ. As shown in Figure 3-4(c), λ_{opt} is 0.4 when β is 0.5, and it is 0.6 when β is 0.7. Thus, a larger λ_{opt} is needed when the rebate is larger with respect to the download cost from the CP's server.

3.10.2 Comparison with Traditional Caching Policies

Figure 3-4(d) shows the cost for Least Recently Used (*LRU*) [32], Least Frequently Used (*LFU*) [136], and Random (*RNDM*) [31] along with those for *Split Cache* when λ set to 0, 1, and λ_{opt}. While *LRU* and *LFU* implicitly leverage object popularity by storing the most popular objects, *RNDM* is insensitive to popularity. As expected, *Split Cache* with λ_{opt} provides the best cost. λ=0 delivers near-best performance for small β. This is because as shown in Eqn. 3-3, for small β (i.e. small rebate C_r), cost depends mainly on

P_M. From Figure 3-4(a), P_M is minimum for λ=0, which corresponds to no-duplication caching.

When β is large (e.g. $\beta \geq 0.7$), λ=1 delivers near-best performance. This is because as shown in Eqn. 3-3, for large β, the cost depends mainly on P_L, which is maximized when λ=1. All traditional policies perform in between *Split Cache* with λ=0 and λ=1. Since *RNDM* is insensitive to popularity, by uniformly distributing the objects in the partition, it is able to increase P_V, which helps it outperform *LRU* and *LFU* for small βs. *LFU*, on the other hand, attempts to distinguish popular objects by keeping track of the number of hits for an object. This explains its performance proximity with *Split Cache* when λ=1.

3.10.3 Partition Object Density

Partition density refers to the number of copies of an object in the *MSWNET* partition. With *Split Cache*, the expected density values are m (number of ECs in the partition) for the duplicated objects, 1 for the unique objects, and 0 for the objects that are not stored in any node. This is confirmed in Figure 3-5(a) which reports densities from simulation for different values of βs. With increasing β, since λ_{opt} increases, more objects are duplicated, thus increasing the density. We show results only up to object-id 50 (i.e. cache size *C*), because objects beyond *C* have only one or zero copy. When $\beta = 0$, since λ_{opt} is also zero, there is no duplication, causing the mC most popular objects to have one copy and the rest of the objects with zero copy.

Figure 3-5(b) depicts simulated object densities for *LFU*, *LRU*, and *RNDM*. Observe that for all three policies, certain amount of density skew (i.e. higher density for more popular objects) is generated by the Zipf based object requests. For *RNDM*, the densities are minimally skewed, since the policy itself is not sensitive to popularities. *LFU*, on the other hand, shows a density pattern that is closest to the *Split Cache* with λ=1 due to its effective sensitivity to object popularity. Similar to the cost results, the density pattern for *LRU* lies in between *RNDM* and *LFU*. This is because the effective sensitivity to object popularity for *LRU* is weaker than *LFU*, but stronger than *RNDM*.

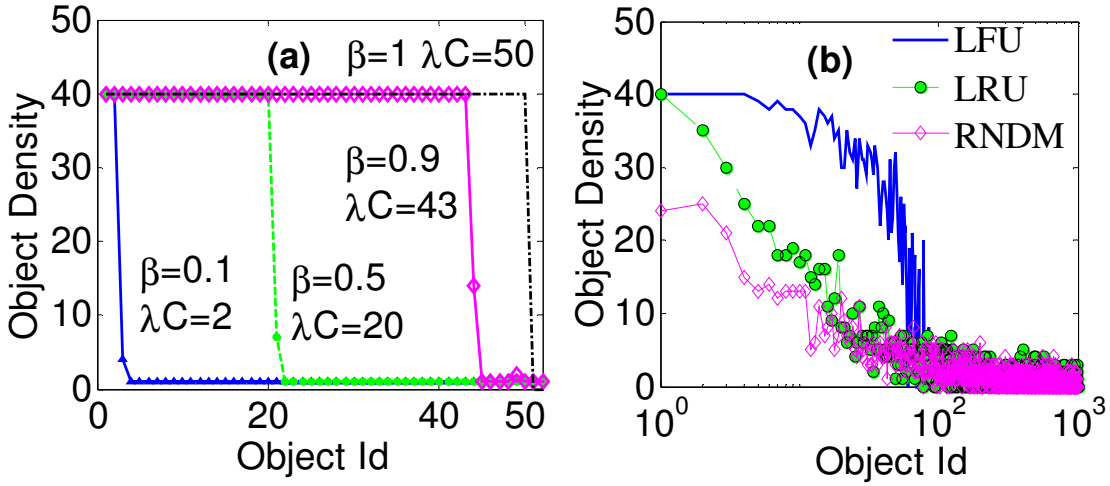


Figure 3-5: Partition object densities

3.10.4 Cost Dynamics over Time

The graphs in Figure 3-6(a,b) demonstrate how the provisioning cost and different hit rates in a static partition (with 40 nodes) changes during the network warm-up phase and also when the object popularities change as a response to external events. As shown in Figure 3-6(a), initially, when all caches are empty, nodes have to download their desired objects directly from the CP's server and therefore the average provisioning cost is very high. This can also be confirmed by Figure 3-6(b) which shows the low local hit rates and the high remote hit rates during the warm-up phase. Gradually when nodes download

objects in their cache, less number of subsequent requests needs to be served through the CP's server, which in turn results in lower costs. The value λ was set to 0.68 which is the optimal value when β is 0.8.

At steady state, each node stores object 1 through objet 34 (i.e. $\lambda.C$) in the duplicate segment of their individual cache, and the remaining 14 slots in the cache are filled with unique objects. At time 30000 seconds, the object popularity profile is altered by swapping the popularities of objects 1 to 10 with those of objects 1000 to 1010. Meaning, object 1000 becomes most popular, and object 1 becomes 1000th popular. A reduction in local hit rate and an increase in remote hit rate can be observed immediately after this alteration in Figure 3-6(b). Due to these changes in hit rates, the provisioning cost suffers from an immediate surge after 30000 second. The algorithm, however, is able to gradually bring the cost down by optimal storing the new set of popular objects in the caches.

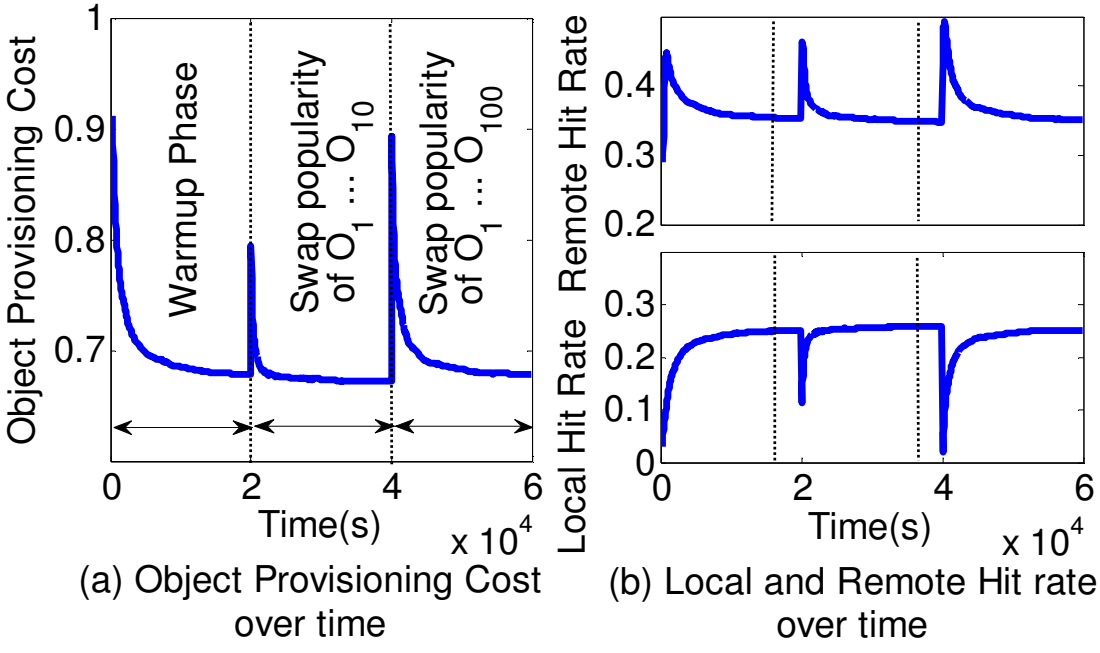


Figure 3-6: Cost and hit rates over time

Figure 3-6(a,b) demonstrate the effects of another popularity alteration that is created again at 40000 second. These dynamics show as to how the proposed cooperative caching can cope with runtime popularity alterations due to external events.

3.11 **Android *SWNET* Test bed**

The *Split Cache* protocol was also implemented as an Android App on a seven-phone Social Wireless Network. Based on Zipf distribution over 5000 objects, each node was programmed to generate 1 request per second. The requests are homogeneous in these experiments. Each phone is able to store up to 50 different objects in its local cache (i.e. $C = 50$). After generating a request for an object, a phone first checks its local cache and if its local search fails, it searches the object in the other six phones using a ad hoc WiFi network acting as the inter-phone peer-to-peer links. If the node does not receive a reply within two seconds after sending the request, it downloads the object directly from a desk-top machine that emulates the CP's server. Note that any object downloaded directly from the CP's server is considered as a unique object and it is stored in the unique area of the cache.

Figure 3-7(a) reports object provisioning costs from both the analytical expressions and from the test-bed when λ varies between 0 and 1, and the rebate to cost ratio β is set to 0.5. The cost is analytically computed according to Eqn. 11 when the parameters m, C, α are set to 7, 50, and 0.8 respectively.

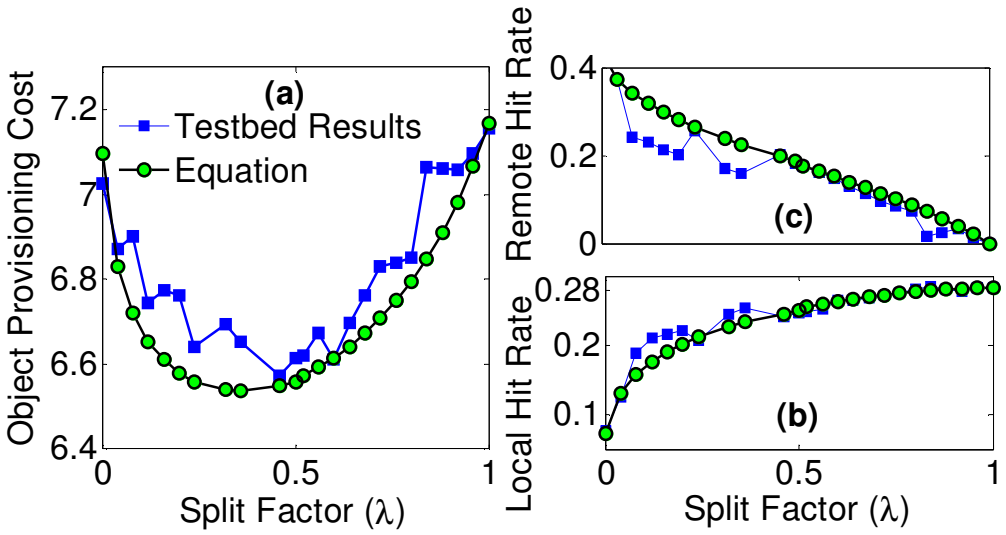


Figure 3-7: (a) Cost and (b,c) Local and Remote Hit Rates

Observe that although the costs obtained from the test-bed maintain values and trends very similar to those from the equation, they are always slightly higher. These differences stem from undesired object duplication as a result of search inaccuracy as follows. When two or more nodes register cache misses at the same time for a supposedly unique object, all of them may attempt to download the object from the CP's server. This can result in undesired object duplications, causing an effective λ which is larger than the target λ_{opt}. Such faulty duplication was also found to happen due to erroneous object search in the events of lost search requests in the WiFi phone network. The impacts of undesired object duplication are higher local hit rates and lower remote hit rates (compared to the equation) and therefore higher provisioning costs.

It should be also observed that the higher costs due to undesired object duplication happens more when λ is small (i.e., $\lambda \ll 1$). This is because when λ is very small, the local hit rate is very low. Thus the number of search requests to the other nodes is quite high. As a result, the absolute number of simultaneous requests and lost search requests

as described above are also high. These cause more frequent erroneous object duplications and subsequently higher cost.

3.12 **Operational Feasibility of Split**

Two relevant operational questions from the Content Provider's standpoint are: a) for a maximum allowable cost point, what is the maximum possible rebate that can be generated for the *MSWNET* users, and b) what would be the minimum cost when the CP attempts to guarantee a pre-specified minimum generated rebate for the *MSWNET* users. In what follows, we address these two questions.

The shaded region in Figure 3-8 envelopes all feasible pairs of cost and generated rebate (GR) that can be generated by varying the parameters β and λ. The top edge of the region is defined by a line representing $\beta = 1$, and λ varying from 1 to 0. Similarly, the bottom edge is defined by a line representing $\lambda = 0$, and β varying from 0 to 1. The figure shows the feasibility lines for specific cases of β, namely, 0, 0.1, 0.5, 0.7, and 1. The feasible region is formed by drawing all continuous β values from 0 to 1.

Point *C* on the feasibility region corresponds to the minimum cost as well as the minimum GR (i.e. zero), which is achieved by setting $\lambda = 0$ and $\beta = 0$. These result into the exclusive caching with zero duplication across the partition, and free object fetching from within the *SWNET*. Point *B* corresponds to the maximum cost as well as the maximum possible GR, which is achieved by setting $\lambda = 0$ and $\beta = 1$. These also result into exclusive caching, but with a hefty rebate C_r, causing the maximum cost and generated rebate. Finally, point *A* on the feasibility region corresponds to $\lambda = 1$ and for any value of β. This is because with full duplication ($\lambda = 1$), there is no need for fetching

an object from the *MSWNET*, causing zero generated rebate. This means that the quantity C_r does not have any impact on cost at this point, leaving the cost independent of β.

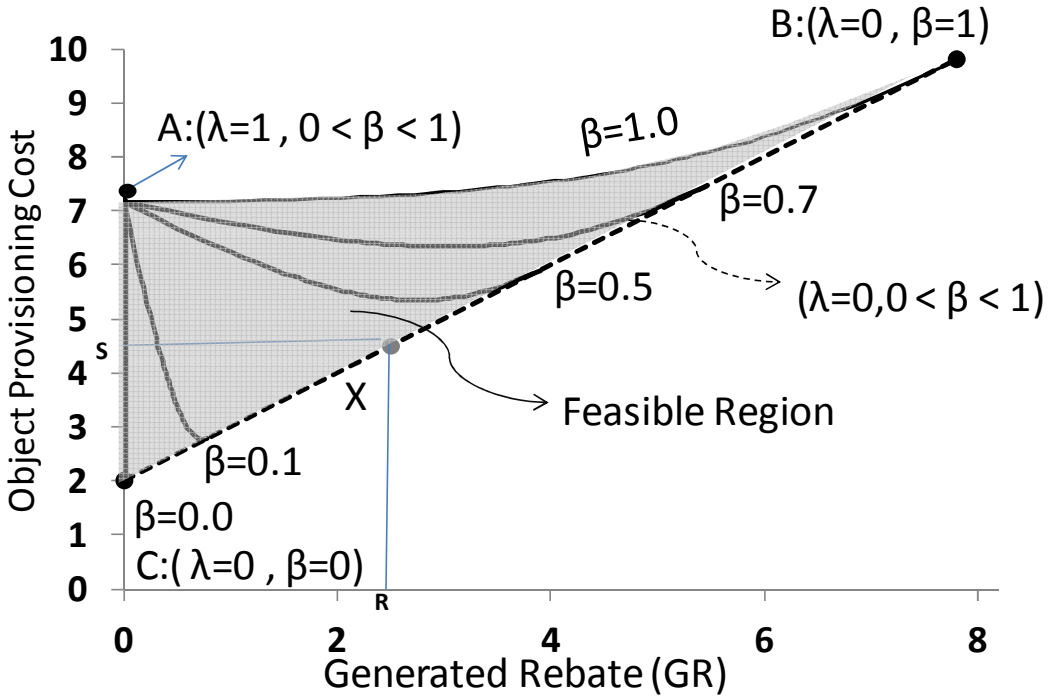


Figure 3-8: Feasible operating region in the cost-rebate plane

The bottom edge (i.e. line *CB*) of the feasible region in Figure 3-8 determines the maximum possible generated rebate for a given maximum allowable cost. For example, if the maximum allowable cost is *S*, then the corresponding maximum GR from the *CB* line is *R*. This situation is shown as the point *X* on line *CB*. The other interpretation of this point is that if the CP wants to provide the minimum GR of *R*, then the minimum possible cost using the Split cooperative caching is *S*. This indicates that the line *CB* also determines the minimum cost when the content provider wants to guarantee a pre-specified minimum GR for the *MSWNET* users.

3.13 **Performance with Non-stationary Networks**

The stationary partition assumption is relaxed in this section. We evaluated *Split Cache* and the traditional policies on a dynamic 98-node *MSWNET* formed by 98 individuals attending the INFOCOM '05 conference [139]. We have extracted the *MSWNET* partition dynamics from a pair-wise interaction trace obtained from [135]. The trace contained synchronized time-stamped pair-wise individual interaction information with a granularity of 4 minutes, which is the Hello packet interval used by a small RF transceiver attached to all 98 individuals while attending the conference. Figure 3-3(a) reports the extracted partition dynamics as the average partition size from individual nodes' perspective. For example, at time 20, average partition size across all nodes is 12.

Figure 3-3 (b) depicts the simulated cost as a function of λ. Observe that the pattern in this graph is exactly the same as that observed for the stationary partition case in Figure 3-10, indicating that the concept of optimal λ also holds for networks with dynamic partitions. Analytical computation of the λ_{opt} in this dynamic case, however, may not be as straightforward due to the wide variation of the partition size as shown in Figure 3-3 (a). A heuristic approach would be to compute λ_{opt} for each node individually based on its own observed average partition size.

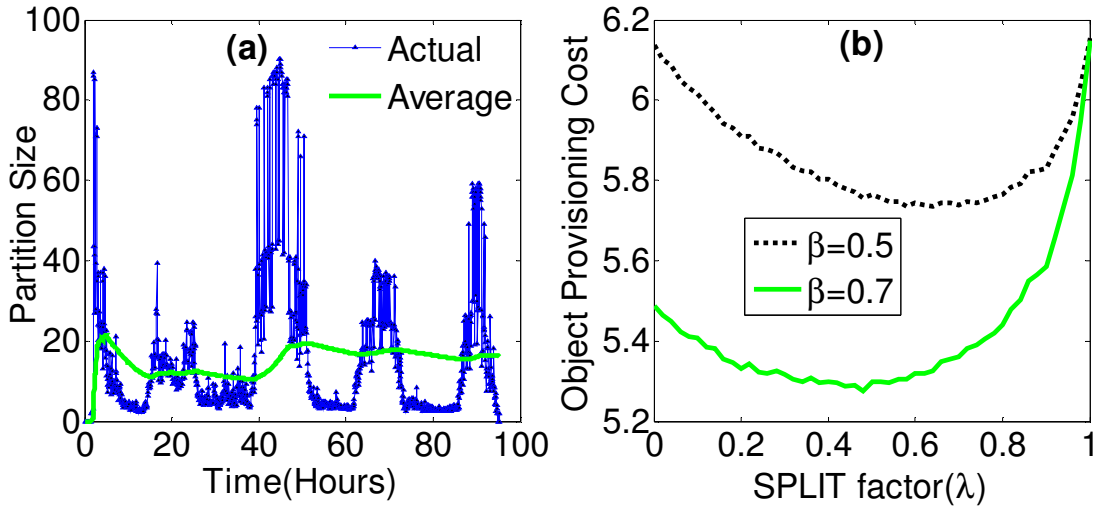


Figure 3-9: Partition dynamics from trace and provisioning cost

Also, unlike in the static case, it is relatively harder to keep consistency of duplication under the dynamic scenario. This is because when a node is in a small partition, it has to download a large number of objects from the CP's server. In other word, from the standpoint of a node, which is in a small partition, those objects are unique. Later, when such a node enters a bigger partition, some of those unique objects may not remain unique anymore in the new partition. To avoid such situations, current partition size is stored along with the object in the cache and during cache replacement objects with smaller partition size are evicted before other objects.

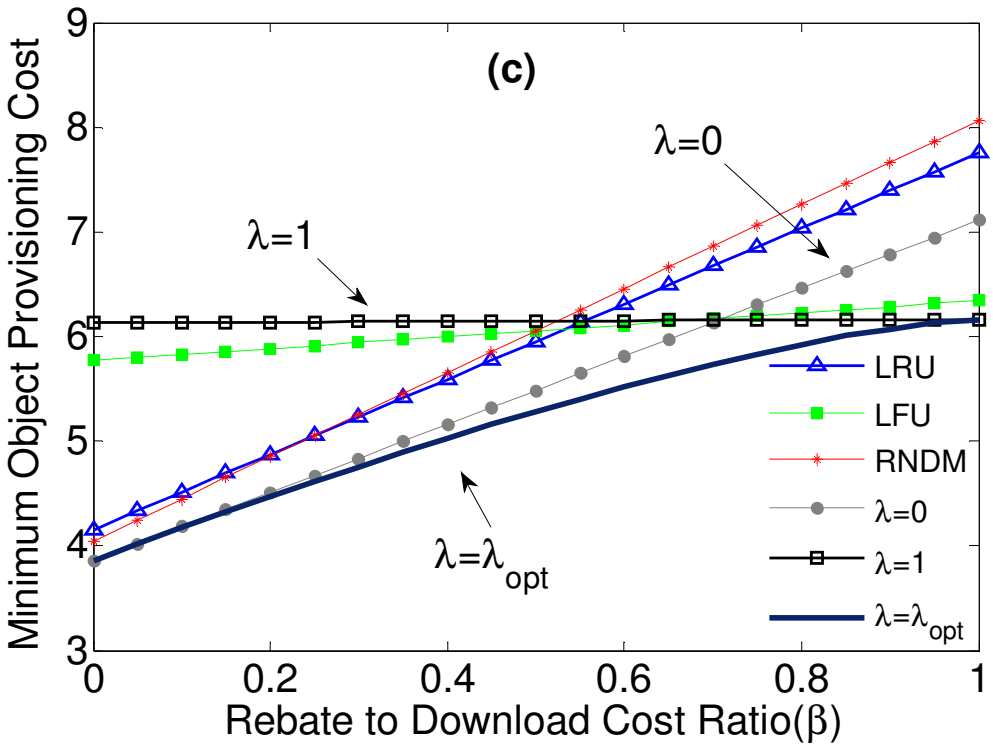


Figure 3-10: Comparative minimum cost

Figure 3-10 depicts that *Split Cache* with parameter λ_{opt} provides the best cost compared to the traditional policies even with dynamic *MSWNET* partitions. Similar to the stationary case, λ=0 and λ=1 deliver near-best performance for small and large β values respectively. Also note that the cost for all policies except *Split Cache* with parameter λ_{opt} grows linearly with β. This is because the quantities P_L and P_V for these policies do not depend on β. Therefore as seen in Eqn. 3-3 the cost is simply a linear function of β. One main difference between the dynamic and stationary network scenarios is that for the dynamic case, *Split Cache* policy with λ=0 outperforms LRU and RNDM for all values of β. This is because as shown in Figure 3-3(a) in some durations of

the experiment, the partition size is quite small. For small partitions, Split Cache with $\lambda=0$ generates the best cost which reduces the average cost of the experiment.

3.14 Summary and Conclusion

In this chapter we developed a cooperative object caching strategy for provisioning cost minimization in Mobile Social Wireless Networks (*MSWNETs*). The key contribution was to demonstrate that the best cooperative caching for provisioning cost reduction requires an optimal split between object duplication and uniqueness. We also analytically derived this optimal split point and subsequently developed the caching performance using a practical network, service and cost formulation that is motivated by Amazon's Kindle e-book delivery model.

Chapter 4 : CACHING FOR HETEROGENEOUS NETWORKS

4.1 Motivation

The *Split Cache* replacement policy with optimal λ minimizes the provisioning cost for stationary networks with homogenous object demands. However, *Split Cache* is not able to minimize the provisioning cost for non-homogenous object requests where nodes have different request rates and request patterns. In this chapter we propose a generalized *benefit* based approach to minimize the object provisioning cost in a network with non-homogenous request model.

4.2 Provisioning Cost with Heterogeneous Requests

Recall that the cost for providing an object is zero when the requested object is found in the local cache. This cost is βC_d when the object is found in the local *MSWNET* partition and C_d when it is downloaded from the CP's server through a CSP's network. The probability that a node '*i*' finds the requested object in its own cache is $\sum_{j \in s_i} p_j^i$ where s_i indicates the set of stored object in node '*i*' and p_i^j shows the probability that a generated request in node '*i*' is for object '*j*'. The probability that a request is found in the network after its local search fails is equal to $\sum_{j \in (S^* - s_i)} p_j^i$ where S^* represents the set of all objects stored in the network. Finally the probability that an object is not available in the network and needs to be downloaded from the CP's server is $1 - \sum_{j \in S^*} p_j^i$. Therefore,

the average provision cost for node '*i*' can be expressed as:

$$Cost_i = \left(\beta \sum_{j\in(S^*-S_i)} p_j^i + \left(1 - \sum_{j\in S^*} p_j^i\right)\right) C_d \quad (4\text{-}1)$$

Average provision cost in the network for all nodes can be calculated as:

$$Cost = \frac{\sum_i \mu_i cost_i}{\sum_i \mu_i} =$$

$$\left(1 - \frac{\sum_i \mu_i \sum_{j\in S^*} p_i^j}{\sum_i \mu_i} + \beta \frac{\sum_i \mu_i \sum_{j\in S^*-s_i} o_i^j}{\sum_i \mu_i}\right) C_d \quad (4\text{-}2)$$

where μ_i shows the request generation rate in node '*i*'. With the heterogeneous request model, the provisioning cost depends on the request rate at each node, object placement in the network, and more importantly, the popularity of each object with respect to each node. Contrary to the homogenous model in which all nodes are interested in the same set of objects with the same popularity distribution, in the heterogonous model the popularity of an object is not the same in different nodes. As a result, finding the optimal object placement that minimizes the provision cost is relatively more complex than that in the homogeneous scenario. In the following section, we propose a distributed algorithm for object replacement which minimizes the overall network-wide cost in the presence of heterogeneous object request patterns.

4.3 **Benefits of Caching**

Suppose Q is the set of nodes that store a copy of object '*j*' in their cache. Let μ_i, be the object request rate for node '*i*' and p_i^j be the probability that a generated request in node '*i*' is for object '*j*' (i.e. node '*i*' generates $\mu_i p_i^j$ requests for object '*j*' per unit time).

The cost of network usage for downloading an object directly from CP's server is C_d. Therefore, storing object '*j*' reduces cost at node '*i*' by the amount $\mu_i p_i^j C_d$ per unit time. This reflects the *benefit* of storing object '*j*' in node '*i*'. Thus the *benefit* of storing object '*j*' in the set of nodes specified by $\mathcal{Q}$ can be written as: $\sum_{\forall i \in \mathcal{Q}} \mu_i p_i^j C_d$

Additionally, every other node in a *SWNET* partition (i.e. nodes that do not store object '*j*' locally) is able to download object '*j*' from one of nodes in $\mathcal{Q}$ with cost βC_d. This reduces the cost of providing object '*j*' to any other node in the network by the amount $(1-\beta)C_d$ for each request for object '*j*'. Total number of requests for object '*j*' by the other nodes in the *SWNET* is equal to $\sum_{\forall k \notin \mathcal{Q}} \mu_k p_k^j$. Therefore, the *remote benefit* of storing a unique object '*j*' in the network is equal to $(1-\beta)C_d \sum_{\forall k \notin \mathcal{Q}} \mu_k p_k^j$. The *total benefit* (the overall amount of cost reduction) of storing a *object* '*j*' in set of nodes specified by '*Q*' can be written as:

$$\sum_{\forall i \in \mathcal{Q}} \mu_i p_i^j C_d + (1-\beta) C_d \sum\nolimits_{\forall k \notin \mathcal{Q}} \mu_k p_k^j$$

This can be rewritten as:

$$(1-\beta) C_d \sum\nolimits_{\forall k} \mu_k p_k^j + \sum\nolimits_{\forall i \in \mathcal{Q}} \beta\, \mu_i p_i^j C_d \tag{4-3}$$

The first term of Eqn. 4-3 refers to the *global benefit* of storing object '*j*' in the network. Note that *global benefit* of storing an object in the network does not depend on the location and the number of copies of that object. The global benefit of objects (*1..N*) can be represented by a vector $\vec{U}$ where

$$U_j = (1-\beta) C_d \sum\nolimits_{\forall k} \mu_k p_k^j. \tag{4-4}$$

The second term of Eqn. 4-3 shows the *local benefits* of storing object 'j' in set of

nodes specified by Q. The *local benefit* of storing object $(1 \ldots N)$ in nodes $(1 \ldots m)$ can be represented by a matrix $D_{m \times N}$ where:

$$D_{ij} = \beta \mu_i p_i^j C_d \tag{4-5}$$

Using the above notations, the *total benefit* of storing object '*j*' in a set of nodes specified by 'Q' can be written as:

$$U_j + \sum_{k \in Q} D_{kj}$$

4.4 **Benefit Based Distributed Caching Heuristics**

With the *Distributed Benefit* based caching strategy presented in this section, when there is not enough space in the cache for accommodating a new object, the existing object with the minimum benefit is identified and replaced with the new object only if the new object shows more total benefit. The benefit of a newly downloaded object is calculated based on its source. When a new object '*j*' is downloaded by node *i* directly from the CP's server using the CSP's 3G/4G connection (i.e. no other copy of the object is present in the *SWNET* partition), the copy is labeled as *primary* and its benefit is equal to $U_j + D_{ij}$.

When the object is downloaded from another node in the *SWNET* partition (i.e. at least one more copy of the object already exists in the partition), the copy is labeled as *secondary* and its benefit is equal to D_{ij}. The new object is cached if its benefit is higher than that of any existing cached object.

In addition to the benefit based object replacement logic as presented above, provisioning cost minimization requires that a primary object within an *SWNET* partition

should be cached in the node that is most likely to generate requests for that object. In other words, a primary object *j* in the partition must be stored in node *i* such that:

$$\mu_i p_i^j > \mu_k p_k^j \; for \; all \; k \neq i \; .$$

To satisfy the above constraint, the *primary* copy of an object '*j*' must always be stored in a node with the highest request generation rate for that object. To enforce this, in addition to the object-ID, a node sends its estimated request generation rate for the requested-object during the search process within *SWNET*. Upon receiving the search request, an object holder compares its own request rate for the object with that of the requesting node. If the request rate of the requesting node is higher and the object copy is a *primary* copy, then the object provider sends the object along with a *change_status* flag to the requesting node. This flag informs the requesting node that the object must be considered as a *primary* copy. Upon receiving of the object and the *change_status* flag, the requesting node considers the object as a *primary* copy and if it can find an object with lower benefit or if it has an empty slot, it stores the new object in its cache. After storing it, the requesting node sends another *change_status* message to the provider node which causes the provider node labels its object as a *secondary* copy. The complete logic of the *Distributed Benefit* heuristics is summarized in Algorithm-2.

Note that in certain rare situations the *object status modification* process fails to satisfy the above constraint. For example, consider a situation in which only one node in the network generates requests and other nodes make no requests. In this case, due to storage limitations, the active node can only store a limited number of objects. The *object status modification* process does not help the active node to offload some objects to the other nodes in the network. Offloading objects to other caches needs extra protocol

syntax and requires additional complexity and overhead in the algorithm and it's beyond the scope of our current work. Object status modification *process* also fails to work perfectly in highly mobile situations. For example, two nodes may consider an object as *primary copy* while they are in the same *SWNET* partition. This may result in storing additional number of copies of some objects. Due to these inconsistencies *Distributed Benefit* heuristics does not guarantee a cost-optimal object placement.

INPUT: O_j //The new downloaded object
flag // Change status flag
IF (*O_j is downloaded from Internet || flag == True***)**
$O_j.benefit = U_j + D_{ij}$
$O_j.label = Primary$
ELSE
$O_j.benefit = D_{ij}$
$O_j.label = Secondary$
END
$O_{min} = Object\ with\ minimum\ benefit$
IF ($O_j.benefit > O_{min}.benefit$**)**
$\boldsymbol{replace}\ O_{min}\ with\ O_j$
$\boldsymbol{send}\ change\ status\ message\ to\ the\ provider\ node$
END

Algorithm-1: A distributed heuristic for object placement in *SWNETs* with heterogeneous content requests in node 'i'

4.5 Performance Upper bound: Optimal Object Placement

In this section we introduce a centralized mechanism in order to find the optimal object placement. First we map the object placement task to a maximum weight matching problem in a bipartite graph. Then we formulate an integer linear objective function to find the maximum weight matching, and we show that the linear programming relaxation of this problem in fact provides the optimal solution.

In a maximum weight bipartite matching problem, for a given bipartite graph $G = (V, E)$ with bipartition $(\mathcal{A}, \mathcal{B})$ and weight function $w: E \rightarrow \mathbb{R}$, the objective is to find a matching of maximum weight where the weight of matching M is given by $w(M) = \sum_{e \in M} w(e)$. Without loss of generality, it can be assumed that G is a complete weighted bipartite graph (zero weight edges can be added as necessary); it can be also assumed that G is balanced, i.e. $|\mathcal{A}| = |\mathcal{B}| = \frac{1}{2}V$, as we can add dummy vertices as necessary.

4.5.1 Optimal Object Placement as a Matching Problem

To map the object placement problem to a *maximum weight bipartite matching*, nodes are modeled by vertices $n_1 \dots n_m$ in partition $\mathcal{A}$, and objects are modeled as vertices in partition $\mathcal{B}$. Initially, we assume that each node is able to store only one object (i.e. cache size is equal to 1) and later we relax this assumption.

In object placement, we may put one object in multiple nodes therefore every object must be modeled by m vertices. For example for object 'j' we create vertices $O_{1j} \dots O_{mj}$ in partition $\mathcal{B}$. A vertex O_{ij} then is connected to the vertex n_i with the weight of D_{ij} which shows the local benefit of storing object 'j' in node 'i'. We also add vertices $Z_{1j} \dots Z_{m-1j}$ in partition $\mathcal{A}$ and connect vertices $O_{1j} \dots O_{mj}$ to them using the edges with weight zero. These new vertices are added to model the situation when object 'j' is not stored in that node. When there is no copy of object 'j' in the network the *global benefit* of object 'j' is lost. To model this situation, vertex G_j is added in partition $\mathcal{A}$ and it is connected with vertices $O_{1j} \dots O_{mj}$ using the edges with weight $-U_j$. Note that there is only $m - 1$ edges with weight zero and therefore, in perfect matching at least one edge

with weight of $-U_j$ must be selected when object 'j' is not stored in any node. The above process is repeated for all objects in the network. Also for every slot of cache space a vertex must be created in partition $\mathcal{A}$ and the whole process of mapping must be repeated again. Figure 4-1 shows a modeled object placement problem when $m = 2, N = 2$ and $C = 1$.

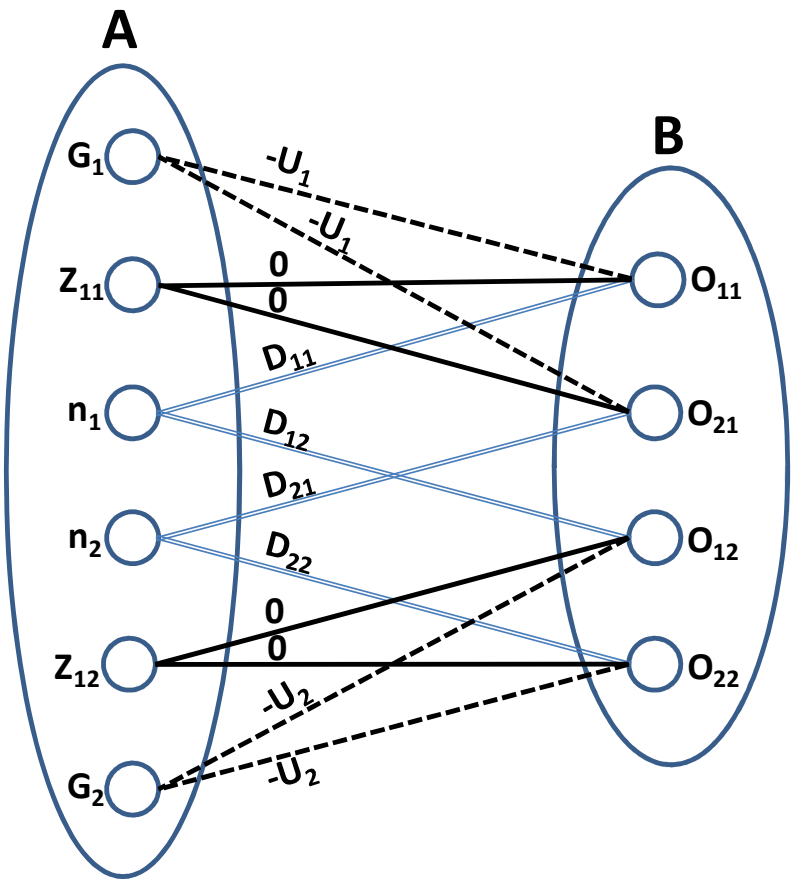


Figure 4-1: An object placement problem as bipartite graph

To make sure all weights are positive, a large enough constant Δ is added to all weights. By adding dummy vertices and edges with weight 0, the graph becomes a complete bipartite graph.

4.5.2 Maximum Weight Matching

For the resulting complete bipartite graph we can formulate *maximum weight perfect matching* as an Integer Linear Programming (ILP) problem as follows:

$$Max \sum_{\forall(i,j)} w_{ij} x_{ij}$$

Subject to: (1) $for\ i \in \mathcal{A} : \sum_j x_{ij} = 1$

(2) $for\ j \in \mathcal{B} : \sum_i x_{ij} = 1$

(3) $x_{ij} \in \{0,1\}\ \ i \in \mathcal{A}, j \in \mathcal{B}$

where $x_{ij} = 1$ if $(i,j) \in matching\ M$ and 0 otherwise. We can relax the integrality constraints by replacing constraint 3 with:

$$x_{ij} \geq 0 \quad i \in \mathcal{A}, j \in \mathcal{B}$$

This gives linear programming relaxation of the above integer program. In a linear program, the variable can take fractional values and therefore there are many feasible solutions to the set of constraints above which do not corresponds to matching. This set of feasible solution forms a *polytope,* and when we optimize a linear constraint over a polytope, the optimum will be attained at one of the "corners" or *extreme points of the polytope* .

In general, the extreme points of a linear program are not guaranteed to have all coordinates integral. In other words, in general there is no guarantee that the solution for linear programming relaxation and the original integer program are the same. However, for matching problem we notice that the constraint matrix of linear program is *totally unimodular* and therefore any *extreme point* of the *polytope* defined by the constraints in linear program is integral. Moreover, if an optimum solution to a linear programming

relaxation is integral, then it must also be an optimum solution to the integer program [140]. Therefore, the solution found by linear programming is optimal for the maximum weight bipartite matching problem to which our object placement problem is mapped into. The *maximum weight matching M* represents the optimal object placement which minimizes provisioning cost in Eqn. 4-2. The optimum result of the linear program can be treated as the upper bound of cooperative caching performance. Such upper bounds are reported in the experimental results in Section 4.7.

The *maximum weight perfect matching* can be also found by Hungarian method (also known as Kuhn–Munkres algorithm) in polynomial time [140][141]. In literature there are many other algorithms for finding the maximum weight perfect matching.

4.6 Evaluation of the Distributed Benefit Strategy

4.6.1 Performance with Homogenous Content Requests

Figure 4-2 depicts average local hit rate, non-overlapping partition hit rate, and miss rate in a network when all nodes run *Distributed Benefit* replacement policy. The results correspond to a 40-node network with homogenous object requests and nodal cache size of 50. As shown in Figure 4-2, all hit rates for *Distributed Benefit* and *Split* Cache with optimal λ are exactly equal for a wide range of rebate-to-download cost ratios. Since it was already proven in Section 3.7 that the *Split* Cache delivers optimal performance under homogeneous object requests, from the observations in Figure 4-2, it can be concluded that the *Distributed Benefit* strategy is also able to deliver optimal performance for the shown range of the rebate-to-download cost ratios. This conclusion is further corroborated in the reported cost numbers in Figure 4-3(a).

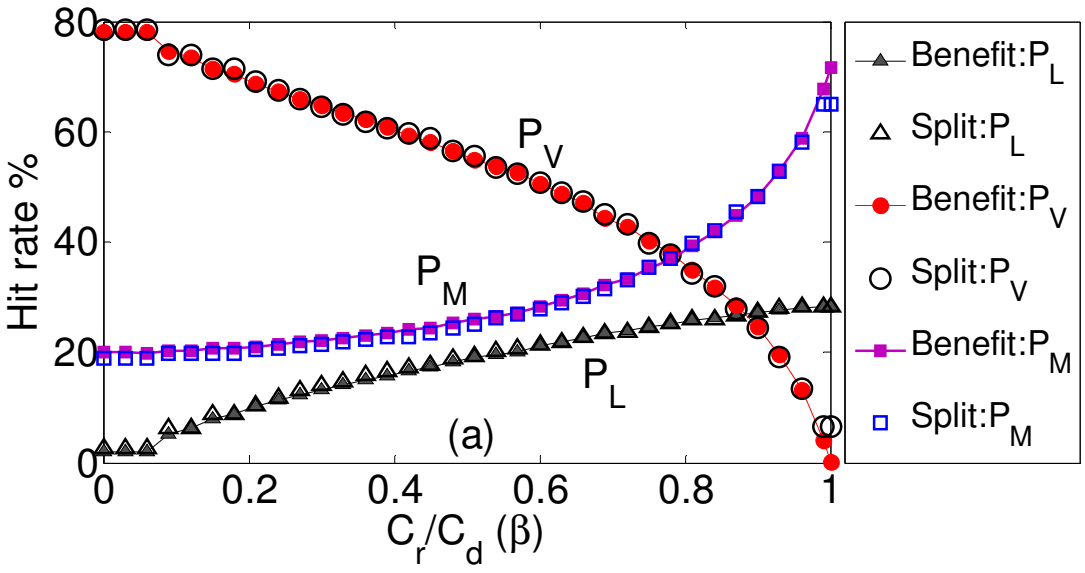


Figure 4-2: Local hit, remote hit and Miss rate for Distributed Benefit and Split Cache with λ_{opt}

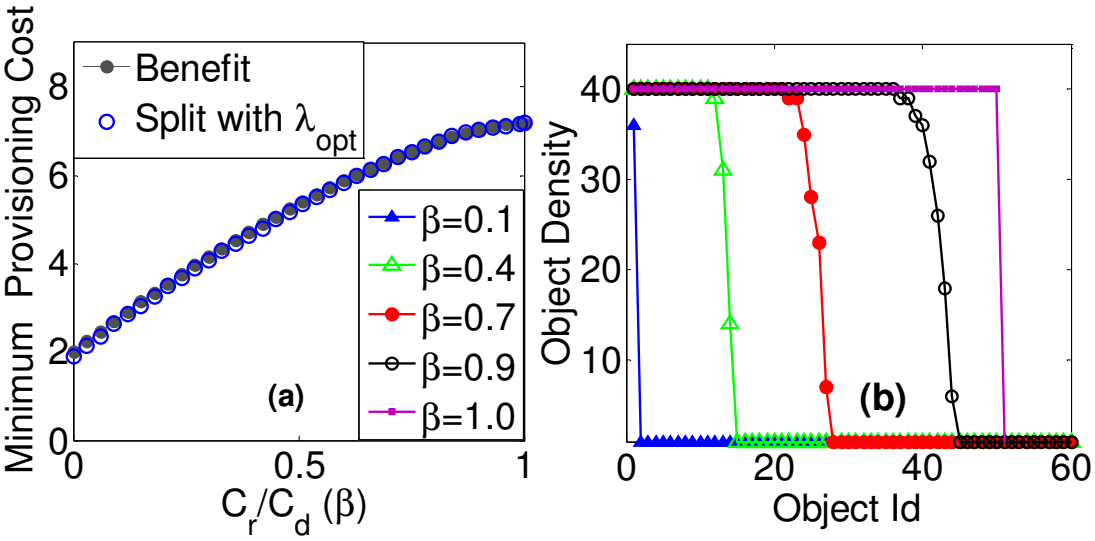


Figure 4-3: (a) Object provisioning cost and (b) object density for Distributed Benefit and Split Cache with λ_{opt}

Figure 4-3(b) demonstrates the object density at steady state within a *MSWNET* partition for the *Distributed Benefit* strategy. It can be observed that the number of copies for different objects follows the optimal object placement as we describe in chapter Chapter 3 above. Furthermore, the least popular duplicated objects in *Distributed*

Benefit and *Split* Cache with optimal λ (Computed using 3-10) are very similar. The small difference is due to randomness and inaccuracy of broadcast search in ns2 simulation. Object density has been measured for different values of β, the rebate-to-download cost ratio. As shown, *Distributed Benefit* keeps less duplicated objects in the partition for small βs. The reason is when β is low, the benefit of duplicated objects is also very low and therefore only few objects are duplicated in the network. To summarize, the results in Figure 4-2 and Figure 4-3 indicate that the *Distributed Benefit* strategy can deliver optimal performance, which is the same performance of *Split* Cache with optimal λ, under homogenous object request patterns.

4.7 Performance with Heterogeneous Object Requests

In this section we study cooperative caching performance when nodes have different request rates and different request patterns. To create node-specific object popularity profiles we have used the following web proxy and web server traces:

BU [142]: The Boston University's proxy trace which contains access information of 28 end users requesting pages from 1840 distinctly different websites during April and May, 1998.

NLANR [143]: A one day trace of HTTP requests to four proxy caches at the National Lab for Applied Network Research, on January 10, 2007. This trace contains access information of 117 end users to 241173 different websites.

For the above two proxy traces we map the web-sites to individual objects, and the users to *MSWNET* mobile devices.

NASA [144]: This trace contains access information of 81983 clients to 21670 webpage of the NASA Kennedy space center web server in Florida during July, 1995.

SASK [145]: This trace contains access information of 162523 clients to 36825 webpage of a web server in the University of Saskatchewan during June to December of 1995. For the NASA and SASK traces we map the web-pages to individual objects, and the clients to *MSWNET* mobile devices.

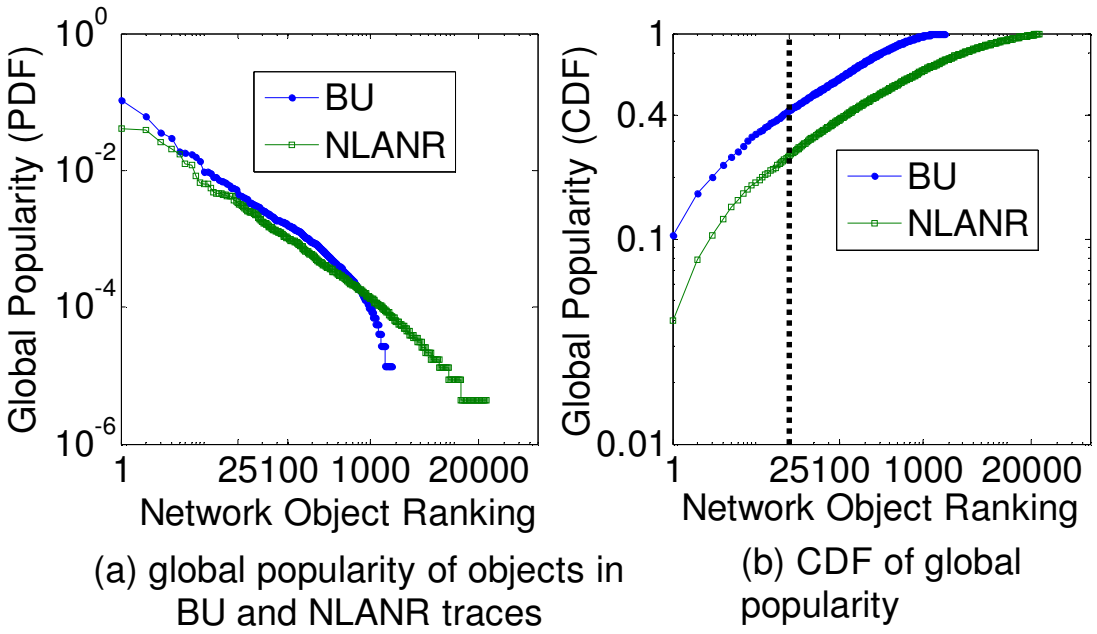
Since the smallest number of clients among all four traces is 28 (i.e. for BU), in order to be able to compare the results across all traces, we extract the access information of 28 nodes with the highest request generation rate from all the trace files and use them in the caching simulation. In all following simulation experiments nodal cache size is set to 25.

Figure 4-4(a) depicts the *global popularity* of objects in BU and NLANR traces. The global popularity of object '*i*' is computed as:

$$global\ popularity_i = \frac{Number\ of\ requests\ in\ the\ network\ for\ object\ 'i'}{The\ total\ number\ of\ requests\ in\ the\ network}$$

It can be observed that the graph in Figure 4-4(a) closely follows a straight line on a log-log scale, indicating the Zipf distribution for object requests.

Figure 4-4(b) depicts the cumulative probability density function of global popularity for both BU and NLANR traces. Observe that when the requests are generated from the BU trace, by storing the first 25 popular objects, each node is able to find 40% of its requested objects in theist local cache. This number is around 20% for requests following the NLANR trace. This confirms that the object popularity in the BU trace is indeed more skewed compared to the NLANR trace.



(a) global popularity of objects in BU and NLANR traces

(b) CDF of global popularity

Figure 4-4: PDF and CDF of global popularity for accessed objects in BU and NLANR

The probability of generating a request for an object in a single node is referred to as the *local popularity* at that node. Similar to the *global popularity*, the *local popularity* also follows a Zipf distribution. However, the set of objects from a single node's standpoint is smaller compared to that in the entire network. The *Local popularity* of object '*i*' at node '*j*' can be computed as:

$$local\ popularity_i^j = \frac{Number\ of\ requests\ by\ node\ 'j'\ for\ object\ 'i'}{The\ total\ number\ of\ requests\ by\ node\ 'j'}$$

The *local popularity* of objects is expected to be different at different nodes. For the BU and NLANR traces, Figure 4-5 (a) depicts the *local popularity* of the most globally popular objects from the standpoint of different individual nodes in the network.

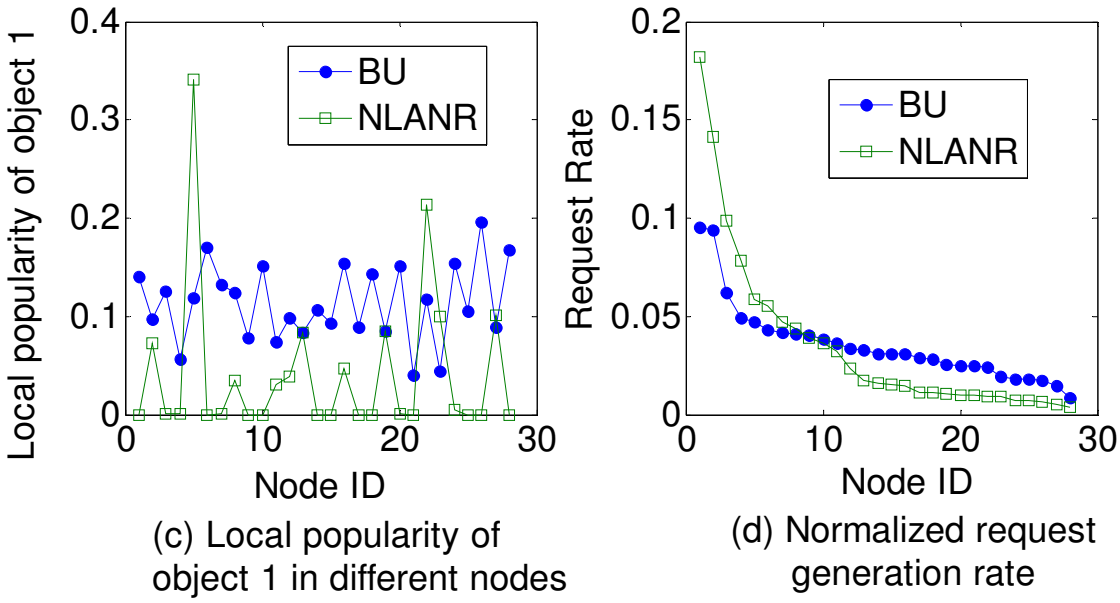


Figure 4-5: Local popularity of the most global popular object and normalized request generation rate in BU and NLANR trace files

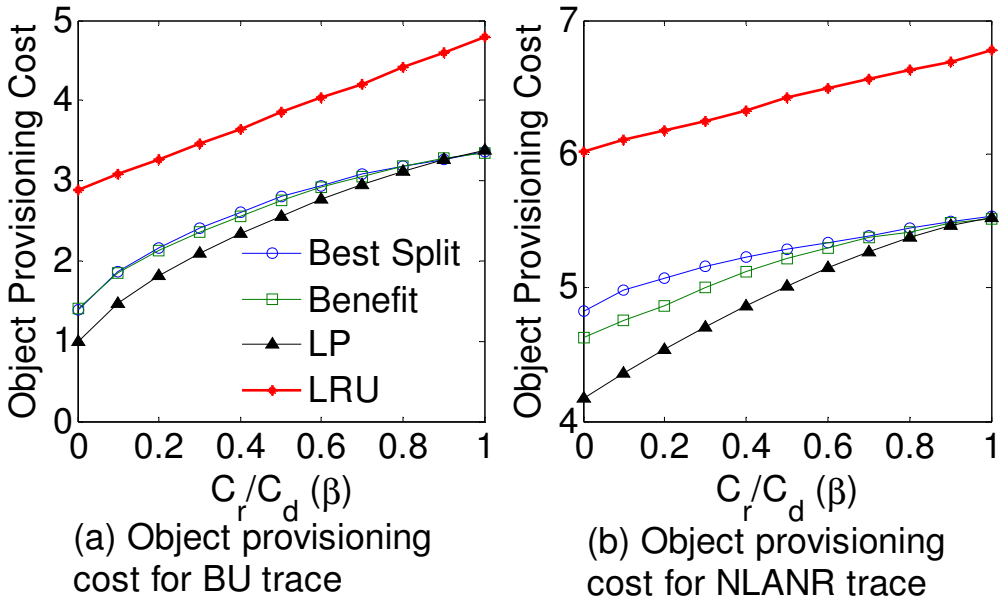


Figure 4-6: Comparing object provisioning cost in benefit based strategy and Split Cache for BU and NLANR traces

Figure 4-5(b) depicts the normalized (by network-wide request rate) request generation rates from different nodes. As shown, few nodes are more active and generate more requests per unit time compared to the others. Individual node-specific request rates

can have a significant impact on the average object provisioning cost, and therefore it is crucial to consider this parameter in object placement algorithm as presented in Algorithm 2 for the *Distributed Benefit* strategy and its associated text. It can be seen that the diversity of request generation rate for NLANR trace is higher than that of the BU trace.

Figure 4-6(a,b) depict the object provisioning cost for *Distributed Benefit, Split Cache*, linear programming (cost lower bound), and traditional LRU with the BU and NLANR traces. Due to the heterogeneous nature of those traces, Eqn.3-10 cannot be used for finding the optimal λ in *Split Cache*. Instead, the optimal λ for *Split Cache* is experimentally found by running the protocol for all possible values of λ, and then selecting the one that generates the minimum cost. This minimum cost is shown as the *Best Split*. Note that LRU is the only representative traditional cache replacement policy for which the results are included in Figure 4-6. This is because it outperformed the other traditional policies, namely, LFU and RNDM.

The following observation can be made from Figure 4-6 (a,b). First, due to optimal object placement, the linear programming has lowest cost compared to those in the other approaches. The cost difference stems mainly from offloading objects from the active nodes (i.e. with very high request generation rate) to other less active nodes as explained in Section 4.3.

Second, the cost in *Distributed Benefit* is always less that with *Best Split* (i.e. *Split Cache* run with the experimentally found optimal λ) and *LRU* replacement policy. The reason is that *Distributed Benefit* attempts, although heuristically, to attain the same object placement goals as by the cost lower bound obtained by *linear programming*. It is

however noted that there exists room for improving the *Benefit Based* heuristics in order to reduce its cost to the lower-bound obtained by the linear programming.

Third, the cost increases with increasing β because by increasing β, the benefit of cooperative caching is reduced. In an extreme case, when β=1, nodes can rely solely on their local cache for reducing the cost. In that case, the performance of *Best Split*, *Distributed Benefit* and *linear programming* become similar.

Fourth, we can also see that for the experiment with the BU trace, *Best Split* and *Distributed Benefit* offer almost the same provisioning cost whereas with the NLANR trace, the difference between the two mechanisms is relatively higher. The reason is that the diversity of request generation rate in the BU trace is less than that in the NLANR trace (see Figure 4-5). Furthermore, the variation of local popularity of objects in NLANR is much higher than that in BU. This is demonstrated in Figure 4-5. To summarize, the lack of diversity in local popularity and request generation rates in the BU trace make this request model perform very similar to the homogeneous case. As a result, the *Split Cache* mechanism is able to provide the same provisioning cost as *Distributed Benefit* whereas in NLANR due to the higher heterogeneity of request generation rates and local popularities, the *Distributed Benefit* heuristics provides better results compared to the *Split Cache* with best performing λ.

Finally, as Figure 4-6 reports, the object provisioning cost for the BU trace is lower than that for the NLANR trace. This can be explained from the graph in Figure 4-4 which shows the cumulative probability density function of popular objects for the BU and the NLANR traces. It can be observed that in the BU trace, storing the same number of objects results in higher hit rates compared to NLANR. In other words, the Zipf

distribution parameter α in BU is higher than that for NLANR, which results in lower provisioning cost for BU. Experiments with the NASA and SASK traces showed performance differences very similar to those between the BU and NLANR traces.

We have done similar analysis of the object provisioning costs with both *Distributed Benefit* and *Split Cache* strategies for the NASA and SASK traces respectively. The performance differences between these two traces were very similar to those between the BU and NLANR traces for the same set of reasons provided with respect to Figure 4-6 (a,b).

It can be observed that the graph in Figure 4-7(a) closely follows a straight line on a log-log scale, indicating the Zipf distribution for object requests.

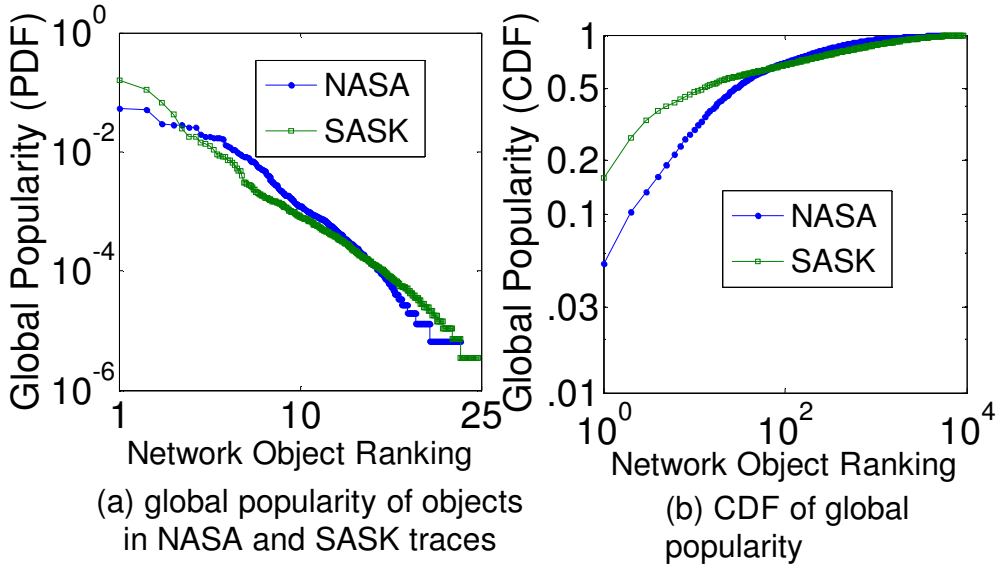
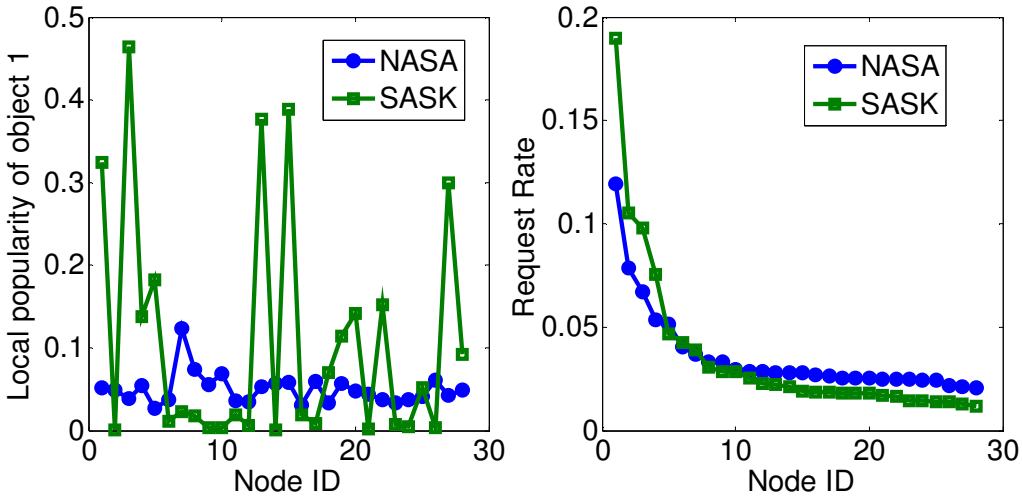


Figure 4-7: PDF and CDF of global popularity for objects in NASA and SASK

Figure 4-7(b) depicts the cumulative probability density function of global popularity for both NASA and SASK traces. Observe that when the requests are generated from the SASK trace, by storing the first 25 popular objects, each node is able to find 50% of its requested objects in theist local cache. This confirms that the object popularity in the

SASK trace is indeed more skewed compared to the NASA trace.

For the NASA and SASK traces, Figure 4-8(a) depicts the *local popularity* of the most globally popular objects from the standpoint of different individual nodes in the network. Figure 4-8 (b) depicts the normalized (by network-wide request rate) request generation rates from different nodes. As shown, few nodes are more active and generate more requests per unit time compared to the others. It can be seen that the diversity of request generation rate for SASK trace is higher than that of the NASA trace.



(c) Local popularity of object 1 in different nodes

(d) Normalized request generation rate

Figure 4-8: Local popularity of the most global popular object and normalized request generation rate in NASA and SASK trace files

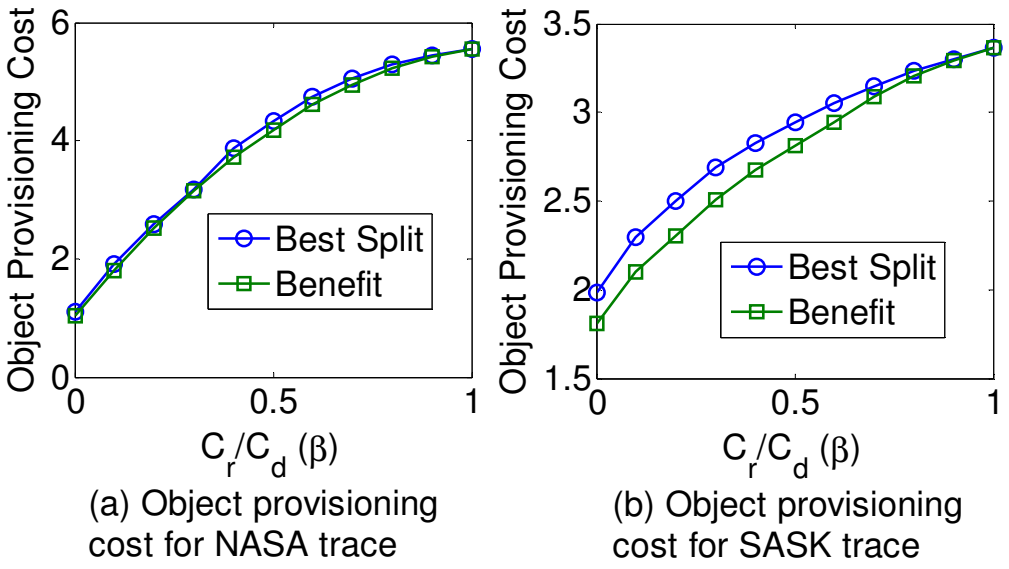


Figure 4-9: Comparing object provisioning cost in benefit based strategy and Split Cache for NASA and SASK traces.

Figure 4-9(a,b) depict the object provisioning cost in both *Distributed Benefit* and *Split Cache* strategies for NASA and SASK traces respectively. For the experiment with the NASA trace, *Best Split* and *Distributed Benefit* offer almost the same provisioning cost whereas with the SASK trace, the difference between the two mechanisms is relatively higher. The reason is that the diversity of request generation rate in the NASA trace is less than that in the SASK trace (see Figure 4-8(b)). Furthermore, the variation of local popularity of objects in SASK is much higher than that in NASA. This is demonstrated in Figure 4-8(a). To summarize the third observation, the lack of diversity in local popularity and request generation rates in the NASA trace make this request model perform very similar to the homogeneous case. As a result, the *Split Cache* mechanism is able to provide near-best provisioning cost whereas in SASK due to the higher heterogeneity of request generation rates and local popularities the *Distributed*

Benefit heuristics provides better results compared to the *Split Cache* with best performing λ.

4.8 Summary and Conclusion

In this chapter we proposed a *benefit based* strategy to minimize the provisioning cost in networks with heterogeneous content demand, in which each node maintains different content request rates and request patterns. It was experimentally shown that the *benefit based* strategy and the cache *split based* strategy can deliver very similar content provisioning costs under homogenous content request model, and the *benefit based* strategy outperforms the split strategy when the content request rates and patterns are heterogeneous across the network nodes.

Chapter 5 : COMMUNITY BASED COOPERATIVE CACHING

5.1 Motivation

A key question for cooperative content caching is how to store contents in devices of mobile user such that the network wide provisioning cost is minimized. In this chapter we show that the optimal caching policy can depend heavily on user interaction patterns and their resulting social community formation. Analysis of real-life human mobility traces [60], [146–148] reveal a strong hierarchical structure in inter-node contact patterns. It is consistently found that certain nodes in a network can belong to strongly connected social communities where they frequently meet each other, while certain other nodes can be isolated and rarely meet other. It is desirable that a node should adjust its cache status based on those in nodes in its same social community. Also, it should not alter its cache based on those outside the community. In this chapter we propose a community based cooperative caching abstraction that exploits community formation hierarchy for reducing network wide content provisioning cost.

5.2 Content Search Model

After an object request is originated, a mobile device first searches its local cache. If the search fails, it searches the object within its local *MSWNET*. Depending on the user tolerable delay (UTD), a device can opt for searching the content in nodes not only within its current physical partition, but also in nodes within its future partitions. This may lead to a larger *Temporal Partition Size* (TPS), thus improving its odds of finding the content. The UTD, which can be content type specific, acts as the time horizon for determining

the TPS.

If the search in *MSWNET* fails, the object is downloaded from the CP's server using CSP's cellular network. In this paper, we have modeled objects such as electronic books, music, etc, which are time non-varying, and therefore cache consistency is not a concern. Each node is assumed to be able to store up to '*C*' different objects, and all objects are popularity tagged by the CP's server [137]. This tag shows the global popularity of an object which is the probability that an arbitrary request in the network is for that object.

5.3 Networks with Community-less Mobility

In this section we relax the assumption in Chapter 3 that all nodes are able to meet each other within their respective temporal partitions. In other words, we consider UTD values that correspond to TPS (temporal partition size) which are smaller than the total number of nodes in the network. Nodes are assumed to follow random walk without any structured contact patterns, thus emulating a community-less mobility model. Contact homogeneity is maintained by ensuring that the probability of contacts between any node pair in the network is the same. Random walk, although not the best model to represent human mobility, is evaluated in this section in order to study the performance of the proposed caching strategy in a community less network.

The local hit rate P_L for a node with random walk mobility depends only on the set of objects stored locally. Following the same logic for developing Eqn. 3-8, the local hit rate in this case can be expressed as $P_L = H_D + H_U/\mathcal{M}$, where $\mathcal{M}$ is the total number of nodes in the entire *MSWNET*. In other words, $\mathcal{M}$ refers to the size of the set of devices that see each other and can cooperate for cost reduction. For example, $\mathcal{M}$ for a student's

device would indicate the size of the set of devices belonging to his/her friends in the University, coworkers in the work place, and social peers in dorm and other frequently visited places. The quantities H_U and H_D used in Eqn. 3-3 need to be recomputed in this case using $\mathcal{M}$ as opposed to m, which was the static partition size.

The remote hit rate with user mobility depends on the Temporal Partition Size (TPS) which indicates the number of distinctly different nodes visited by a node within a time horizon. For a given user mobility pattern and a pre-defined time horizon, let Γ $(1 \leq \Gamma \leq \mathcal{M})$ be the TPS observed by a node at an arbitrarily chosen point in time. If $Q(\Gamma)$ is the probability that a node's TPS at an arbitrarily chosen time is Γ, and P_V^{Γ} represents the remote hit rate within a static partition size Γ, then P_V in the presence of mobility can be written as:

$$P_V = E(P_V|\Gamma) = \sum_{\Gamma=1}^{\mathcal{M}} Q(\Gamma)P_V^{\Gamma} = \sum_{\Gamma=1}^{\mathcal{M}} H_U\,(\Gamma - 1)Q(\Gamma)/\mathcal{M}$$

The above equation can be further simplified as:

$$P_V = (E(\Gamma) - 1)H_U/\mathcal{M}\,.\mathrm{P_V} = \frac{\mathrm{E(\Gamma)\text{-}1}}{\mathrm{m}}\mathrm{P_V} = \frac{\mathrm{E(\Gamma)\text{-}1}}{\mathrm{m}} \tag{5-1}$$

Substituting the above expressions of P_L and P_V for the mobility case in Eqn. 5-3, the average provisioning cost in the presence of mobility can be calculated as:

$$Cost = \left(1 - \left(\frac{(1-\beta)E(\Gamma)+\beta}{\mathcal{M}}\right)H_U - H_D\right)C_d. \tag{5-2}$$

After expanding H_U and H_D, Eqn. 5-4 can be written as a function of λ. By equating the cost derivative to zero, we can compute λ_{opt} at which the cost is minimized.

5.4 Networks with Community based Mobility

Unlike in random walk, real-life human mobility is often abstracted by underlying social community structures, which can and should be leveraged for cooperative caching. Node contact probabilities in community based mobility can have various forms of locality which should be considered while designing caching policies. In this section we develop a hierarchical split-cache strategy for leveraging such locality structures in inter-contact times.

5.4.1 Hierarchical Split Caching

From each node's standpoint, any other node in a mobile network can be in-community or out-of-community (i.e. *stranger*). Nodes in the same community meet more frequently an often for longer duration. Frequencies of contacts between strangers are usually low and last shorter. These properties are experimentally found in a number of real-life human mobility traces [62], [63] and models [60], [149] used by various researchers in the community.

These observations lead to the following approach in which the level of cache collaboration between two nodes depends on whether they are in-community or out-of-community. The split-cache mechanism is extended to a hierarchical version (Hierarchical Split Cache or HSC) in which the cache space in each node is divided into three separate areas. Certain number of very objects is cached in the *first* area in order to secure their access locally. Objects stored in this area can be duplicated at nodes across and outside communities. The *second* area is reserved for cooperation among in-community nodes. The objects stored in this *second* cache area of a node cannot be duplicated across its in-community member nodes, but duplication of such objects can be

allowed in its stranger nodes. In other words, these are unique within a community. Finally, the *third* area in the cache is used for implementing global cooperation among all nodes by maintaining global uniqueness. This *third* area of cache helps nodes in a small community to take advantage of cooperation with stranger nodes. This is especially useful in situations in which one meets many strangers in public places such as train stations and airports. One's device in such situations can retrieve content from the *third* cache area of those strangers' devices. Figure 5-1 illustrates the hierarchical splitting of the cache space. The best split factors between the three areas need to be determined based on the social community properties of the network.

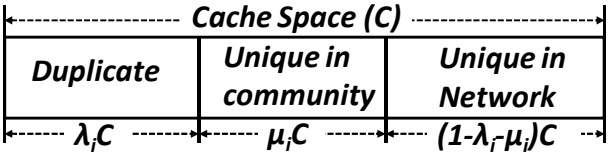


Figure 5-1: Hierarchical partitioning for supporting communities

Cost Computation: The average local hit rate for nodes in the i^{th} community can be computed as:

$$L_i = f(\lambda_i C) + \frac{f((m_i\mu_i + \lambda_i)C) - f(\lambda_i C)}{m_i} + \frac{C(1 - \lambda_i - \mu_i)}{N_u} \times H_u$$

where m_i is community size, $N_u$is the total number of unique objects stored in the *third* cache area of all network nodes, and H_u is the corresponding hit rates of these N_u objects. N_u can be computed using the following equation:

$$N_u = \sum_{\forall i} m_i(1 - \lambda_i - \mu_i)C$$

Hit rate of unique objects can be written as:

$$H_u = f(\mathrm{O}_{\max} + N_u) - f(\mathrm{O}_{\max})$$

where the parameter O_{max} is the least popular among all objects that have been stored in the *second* cache area. It can be computed as:

$$\mathrm{O}_{\max} = \max_{i}\big((\mu_i - \lambda_i)m_i + \lambda_i\big)C$$

Let ϕ be the contact probability between in-community nodes with a specified user tolerable delay (UTD). Then the percentage of requests that can be found in remote caches within the i^{th} community can be computed as:

$$R_{ic} = (m_i - 1) \times \phi \times \left(\frac{\left(f\left((m_i\mu_i+\lambda_i)C\right)-f(\lambda_i C)\right)}{m_i} + \frac{C(1-\lambda_i-\mu_i)}{N_u} \times H_u \right) \quad (5\text{-}3)$$

The first term refers to hit rate of objects stored in the *second* cache area of nodes in the i^{th} community, and the second term indicates hit rate of objects stored in the *third* cache area.

A node in i^{th} community may also meet strangers from other communities. The average remote hit rate of objects found in the stranger nodes can be computed as:

$$R_{is} \approx \frac{\Gamma_i}{M} H_u \quad (5\text{-}4)$$

where Γ_i represents the average number of stranger nodes that an i^{th} community node is expected to see during a specified user tolerable delay. The average object provisioning cost for an i^{th} community node can be calculated as:

$$Cost_i = (1 - (1 - \beta)R_i - L_i)C_d$$

where $R_i = R_{ic} + R_{is}$ is the total remote hit rate for nodes in community-*i*. The overall cost can be computed as:

$$Cost = \frac{\sum_{\forall i} m_i Cost_i}{M} \quad (5\text{-}5)$$

where M is the total number of nodes in the entire network. To minimize the overall cost, we need to differentiate the above function with respect to all unknown parameters (i.e. $\lambda_i and\ \mu_i$) and equate it to zero.

Observe that when the communities are completely independent (i.e. when nodes never meet strangers from other communities), the overall cost can be minimized by simply minimizing the cost in each community. In this case, the problem reduces to the stationary partition case, as in Chapter 3, with a single cache split parameter for each community.

5.4.2 Centrality Based Community Detection Algorithms

In order to deploy hierarchical split caching as described in Section 5.4.1, a community detection algorithm is needed using which nodes can identify other members of its community. As a first step, the contact pattern of mobile users is mapped as a weighted graph in which the vertices represent mobile nodes and edges represent connections between nodes. The weight of an edge indicates the strength of connection between two nodes. For example, if two nodes always move together (i.e. they are always connected), the weight of the edge between them will be 1. In the other extreme, two nodes that never meet should have an edge with weight zero.

As defined in the content search model in Section 5.2, a node may opt to wait up to a preset user tolerable delay for searching content in nodes not only within its current physical partition, but also in nodes within its future partitions. The weight of an edge between any pair of nodes must be computed based on the preset UTD value. Such weight is defined as the probability that those nodes meet within UTD duration. In this section, we review three commonly used community detection algorithms from the

literature that we have used for evaluating our proposed hierarchical cache split mechanism.

Girvan-Newman [150], [151] is one of the oldest hierarchically divisible community detection algorithm that works based on finding and removing network edges with high *betweenness*. Edge *betweenness* is defined as the number of shortest paths between pairs of vertices that pass through it. When there is more than one shortest path between a pair of vertices, each such path is given equal weight such that the total cumulative weight of all the paths is one. The Girvan-Newman algorithm works well when the communities are loosely connected by few inter-group edges. In this case, all shortest paths between the communities must pass through one of those few inter-group edges. Thus, the edges connecting the communities will have high edge *betweenness*, and by removing them the groups (i.e. communities) can get separated.

The procedure of finding and removing edges stops when the cumulative *modularity* of all communities in the resulting network is maximized. Modularity of a community is defined as the actual number of edges within the community minus the expected number of edges in an equivalent network (with the same community divisions) when edges placed at random. The modularity can be used as a metric that shows strength of community structure in a network. For a random graph this quantity is close to zero which indicates no community structure. In practice, modularity of social based networks falls in the range from 0.3 to 0.7.

The original algorithm of Girvan-Newman has a complexity of $\mathcal{O}(N^3)$. The authors in [152] proposed a greedy modularity optimization which is a fast implementation of Girvan-Newman algorithm. Using more efficient data structure they reduce the

complexity of the algorithm to $\mathcal{O}(Nlog^2N)$. In this paper we term this algorithm as ***fastcommunity*** in the experimental results in Section 5.5.

Blondel et al. [153] propose a two-phase heuristic algorithm which is based on a local optimization of Girman-Newman modularity. As a first step, each network node is assigned to a different community and then the algorithm attempts to increase the cumulative network modularity by combining nodes and their neighbors. A node is grouped with other nodes that result in a network with increased modularity. The process of grouping stops when a local maximum of the network modularity is attained, i.e. when no individual grouping move can improve the modularity any further. In the experimental results in Section 5.5, this fast community detection algorithm is labeled as ***fastfold***.

There are many **hierarchical clustering algorithms** that group *similar* nodes within communities. The similarity between two nodes can be defined based on criteria such as Euclidean distance, Pearson correlation [154] and cosine similarity. In this chapter, we use Pearson correlation to find node similarities which are computed as:

$$x_{ij} = \frac{\frac{1}{n}\sum_k (A_{ik} - \mu_i)(A_{jk} - \mu_j)}{\sigma_i \sigma_j}$$

where $\mu_i = \frac{1}{n}\sum_j A_{ij}$, $\sigma_i^2 = \frac{1}{n}\sum_j \left(A_{ij} - \mu_i\right)^2$.

With Pearson correlation measure, nodes with more of common neighbors have higher similarity. A stopping criterion is used in which no clique bigger than 3 nodes remains in the entire network. In the results Section 5.5, we label the results of such community detection mechanism as ***clique***.

5.5 Evaluation of Community-less Mobility

5.5.1 Temporal Partition Characterization

This section reports the characterization of partition dynamics in an *MSWNET* with simulated homogeneous human mobility within a campus-like setting. 40 mobile nodes with 100m transmission range were simulated in ns2 using random waypoint mobility model with an average speed of 1-2 m/s and pause time (between movements) of 100 sec.

Figure 5-2 depicts the TPS distribution when the 40 nodes are allowed to move within an area of 1500m x 1500m. The node-to-node connectivity is detected using beacon messages sent from each node with a period of 500 msec. The four graphs in Figure 5-2 reports TPS distribution with the User Tolerable Delay (UTD) set to 1s, 120s, 210s, and 600s respectively. The UTD represents the delay that an end consumer is willing to tolerate before a requested content is provisioned into his or her device. In other words, UTD determines the time horizon as explained in Section 5.2. As expected, with a larger UTD, a node is able to see larger number of distinct nodes within the time horizon (i.e. UTD), causing a larger TPS. This explains the right-shift in the distribution with increasing UTD. For the largest case (i.e. 600s) a node is able to see almost all other nodes, resulting in the large peaks near 40. With very small UTD (e.g. 1s), a node remains either isolated most of the time, or sees partitions of very small sizes, resulting in the large peaks near 1.

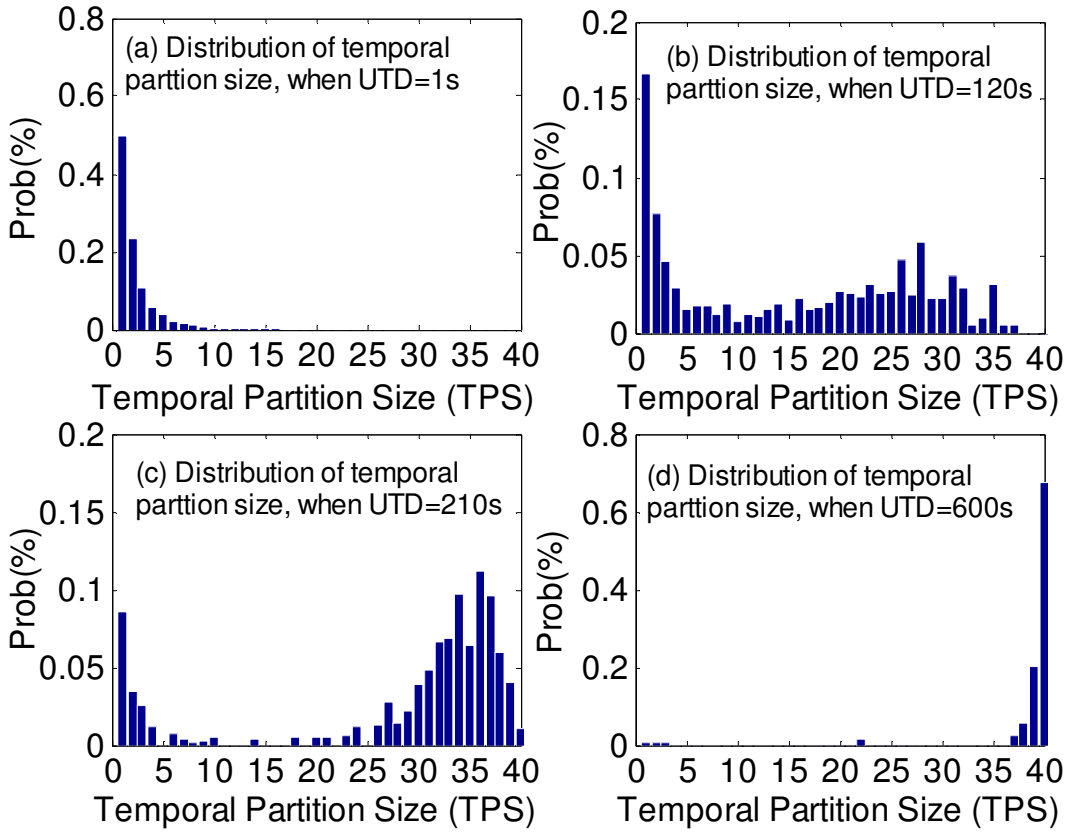


Figure 5-2: Temporal Partition Size (TPS) distribution

Figure 5-3 summarizes the impacts of UTD and also spatial node density on the average TPS, the trends in which are consistent with the distribution peak shifts observed in Figure 5-2.

As shown in the above results, the effects of user mobility in a mobile *MSWNET* can be captured in terms of the temporal partition size distribution as observed by a participating node. In the equations in Section 5.3, it was shown as to how those distributions can be used for analyzing the effects of mobility on cooperative caching performance. Since both user tolerable delay and *MSWNET* expanse (i.e. area) affect the user mobility in a very similar way, for all the following analysis we will use only UTD as the representative parameter for mobility.

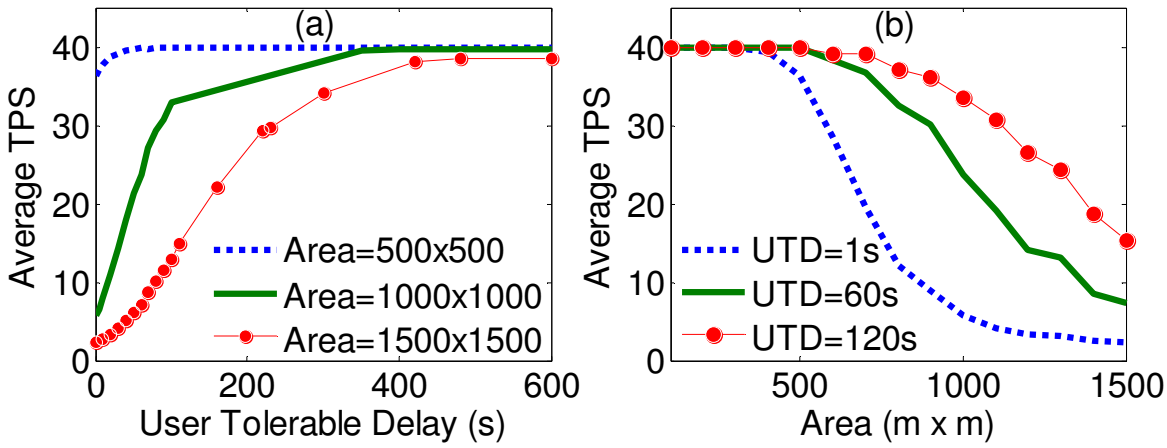


Figure 5-3: Average TPS with different UTDs and node densities

5.6 Hit Rates and Costs under Community-less Mobility

Figure 5-4(a) shows the network wide average hit and miss rates for the mobility scenario corresponding to the TPS distribution in Figure 5-2(a) (i.e. UTD of 1s in a 1200m x 1200m network). Since the average TPS in this case is very small (i.e. around 2) a node can rely on only another node for cooperation. This explains the extremely low remote hit rate (P_V) in Figure 5-4(a). As for local hit rate (P_L), it depends primarily on the set of locally stored objects, which is an increasing function of the split-factor λ. As shown in Figure 5-4(a), with large λ since more objects are duplicated, the local hits are more likely. Since the remote hit rates in this case are negligible, the miss rates (P_M) are effectively the complement of the local hit rates. That explains the decreasing trend of P_M with increasing λ.

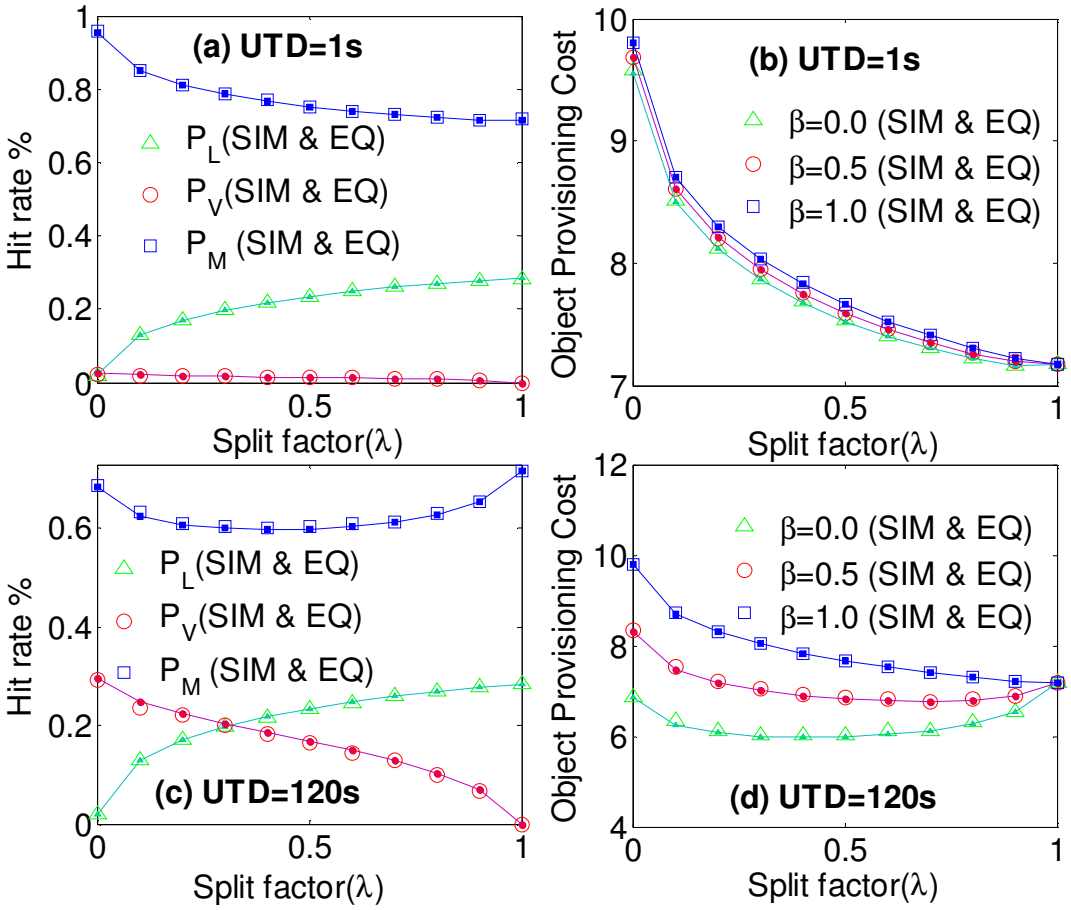


Figure 5-4: Hit rates and cost: (a,b) UTD 1s; (c,d) UTD 120s

Figure 5-4(b) demonstrates the network wide average object provisioning cost for the same mobility scenario for three different values of the rebate-to-download-cost ratio β. In this case, with negligible PV the cost expression from Eqn. 4 reduces to $Cost = (1 - P_L)C_d$.This shows that the cost can be minimized by maximizing the local hit rate P_L, which can be achieved by setting the split factor λ to 1. Intuitively, in very small partitions (e.g. 2 in this mobility scenario), a node should rely mainly on its own cache by storing the maximum number of duplicated popular objects. This corresponds to the value of λ to be 1.

Figure 5-4(c) shows the hit and miss rates for a different mobility scenario corresponding to the TPS distribution in Figure 5-4(b) (with UTD set to 120s in the same 1200m x 1200m network). As indicated by the distribution, since the users are willing to tolerate a large delay of 2 minutes, a node's average temporal partition size in this case is larger (i.e. an average of 15) than the one used for the results in Figure 5-4(a,b). Having access to more nodes in their temporal partitions, the nodes in this scenario enjoy higher remote hit rates compared to the previous mobility scenario. With larger split factor λ, since a larger portion of each node's cache is used for duplicated objects, the unique object count in a temporal partition reduces. This causes the observed reduction in remote hit rate with increasing λ in Figure 5-4(c). From the P_L values in Figure 5-4(a,c), it can be confirmed that the local hit rate has very little sensitivity on mobility and its resulting temporal partitions. Generally, the miss rate P_M decreases with increasing P_L, and increases with decreasing with increasing P_V. This explains as to why with increasing λ, P_M initially reduces when P_L is high, and then increases later when P_V is low.

The graph in Figure 5-4(d) indicates that the cost for very small β (i.e. 0) follow a trend (i.e. lowest for an optimal λ of 0.4) which is similar to that in static networks as observed in Figure 3-4. In a static network, for $\beta = 0$, the cost is minimized by avoiding object duplications by setting λ to 0. However, with mobility, since a node can rely only on a subset (i.e. the TPS) of all the network nodes, setting λ to 0 is not the best solution. Certain amount of duplication and the resulting local hits are needed for minimizing the cost. The role of local hit rate becomes more important as β increases, because the benefit

of finding an object in the remote caches within the TPS becomes less. This explains why optimal λ shifts to the right with increasing β. In the extreme case, when $\beta = 1$, cooperative caching is no longer useful. The nodes in that case need to rely only on the local hits. Note that the close agreements between the analytical and simulation results in Figure 5-4 validate the Eqns. in Section 5.3.

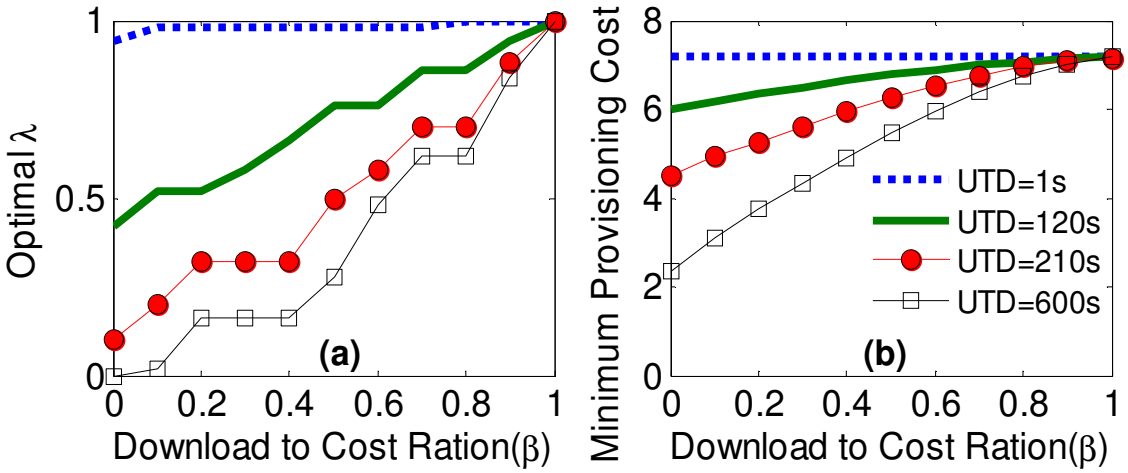


Figure 5-5: Optimal λ and minimum cost for different UTDs

Figure 5-5(a) reports the optimal split factor for various β and user tolerable delays representing varying temporal partition distributions. With lower UTDs, the nodes are more likely to be isolated, and they need to rely primarily on their locally duplicated cached objects. This explains why optimal λ is close to 1 (i.e. allowing maximum duplications) when the user tolerable delay is set to 1s.

For a given β, with larger UTDs, the nodes are increasingly able to rely more on remote hits within larger temporal partitions, leading to larger optimal λ values. For a given UTD, as the quantity β increases (i.e. the cost of cooperation increases), the *Split Cache* policy attempts to increase local duplications in each node. This results in larger optimal λ values. This trend can be observed for all UTDs in Figure 5-5(a).

The graph in Figure 5-5(b) depicts that the minimum provisioning cost can be lowered by reducing the quantity β and/or by increasing the user tolerable delay (i.e. larger temporal partition size). As shown in the Figure 5-5(b), the absolute minimum cost can be achieved by eliminating the rebate and setting a large UTD of 10min.

5.7 Evaluation with Community Based Mobility

5.7.1 Simulation Setup and Mobility Traces

The performance of proposed caching has been evaluated with two human mobility traces (i.e. Infocom '06 trace [147] and UCSD trace[62]) and a synthetic human mobility generator HCMM [60]. HCMM is an extension of community based mobility model (CMM) which works based on the Caveman-model proposed in [148].

In the UCSD [62] project 275 freshman were equipped with WiFi enabled PDA devices that periodically scanned and logged the visible access points for a period of 77 days. The contacts between nodes, determined by access point sharing, were extracted from the log file. The average inter-contact time in this trace was about 19 hours. Figure 5-6(a) reports the number of associated users per hour across all access points in the campus during the first 14 days of the experiment.

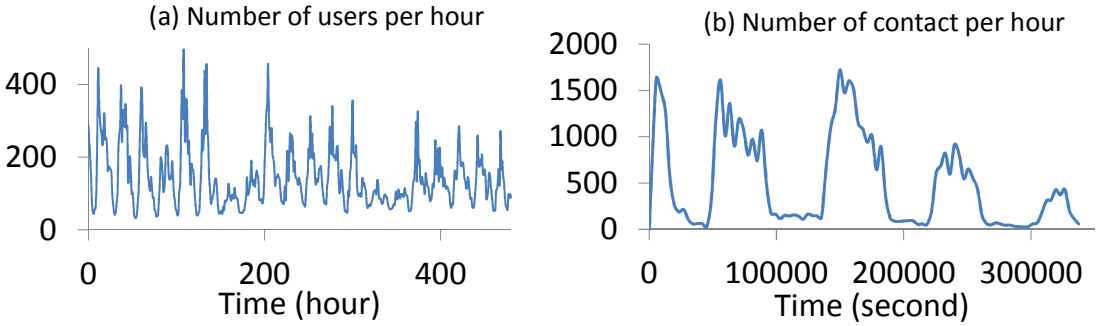


Figure 5-6: Number of associated users per hour in the UCSD trace; (b) Number of contacts per hour in Infocom '06 trace

The Infocom '06 trace includes records of Bluetooth sightings by a group of 78 users attending Infocom 2006. Each of the 78 users was carrying a small iMote device during the conference. 5-6(b) reports the number of contacts between nodes per hour during 4 days of experiment. Bluethooth discovery is done with a frequency of once every 4 minutes.

For all trace files, a proximity matrix is generated based on the number of recorded contacts between users. To compute proximity matrix, the trace is first divided to a series of UTD long intervals. The proximity between node *i* and *j* is then computed based the number of intervals that these nodes had a contact. For example, if the total number of intervals in a trace is *I* and nodes *i* and *j* have meeting in *K* of the *I* intervals, then the proximity between *i* and *j* is equal to *K/I*. The proximity matrix indicates the strength of connections among all participating nodes. The set of pairs that have proximity higher than a specific threshold are fed into the community detection algorithms described in Section 5.4. The results reported in this section are based on *fastcommunity* detection algorithm.

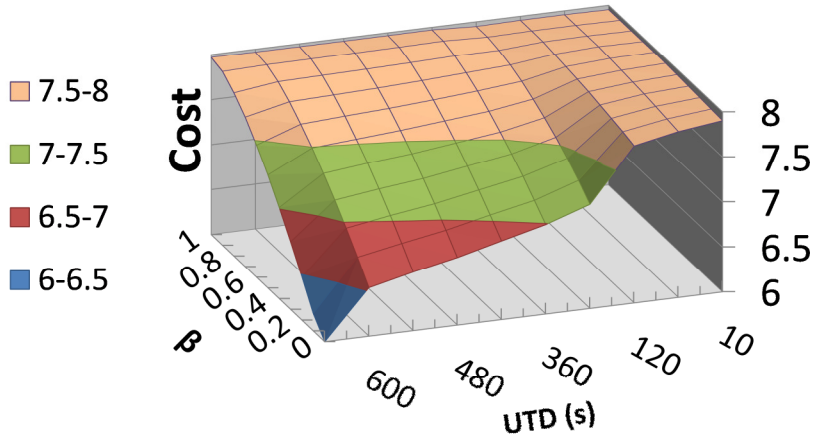


Figure 5-7: Object provisioning cost for HCMM trace

5.7.2 Mobile Social Wireless Networks

HCMM trace: Figure 5-7 reports the measured object provisioning cost for a mobile network of 50 nodes generated using HCMM model [60]. With increasing user tolerable delay, each node is able to meet more nodes and therefore, the object provisioning cost reduces. This is especially pronounced when the rebate to download cost ratio (i.e. β) is small. In extreme case, when $\beta=0$ (i.e. downloading objects from other nodes involves no cost), the provisioning cost can be as low as 6 per object when UTD is 1200 seconds. It goes up to 7.8 when the UTD reduces to 10 seconds. The reasons for lower cost with higher UTDs are as follows.

First, with increasing UTD the percentage of isolated nodes reduces due to increasing community size. Isolated nodes are nodes that do not belong to any community, thus unable to leverage cooperative caching, leading to higher provisioning costs. Thus by reducing the number of isolated nodes it is possible to reduce the average network-wide cost. The second reason for cost reduction with higher UTD is due to higher community cohesions. Cohesion factor is defined as the ratio of number of edges between nodes in a community to the number of edges in a fully connected community with the same size. This metric is defined to measure the strength of connections in a community. By increasing UTD, the connection between nodes within a community can become stronger. This let nodes to have more cooperation, which in turn can reduce the provisioning cost.

Infocom 06 trace: For the infocom '06 trace there were very few contacts between nodes during the night time part of it. Due to this reason, we have first removed the night time part from the trace in order to avoid any skew in the results. The proximity matrix is

then computed based on the reduced trace. From the proximity matrix, all pairs with proximity higher than a threshold are extracted and fed into the *fastcommunity* detection algorithm.

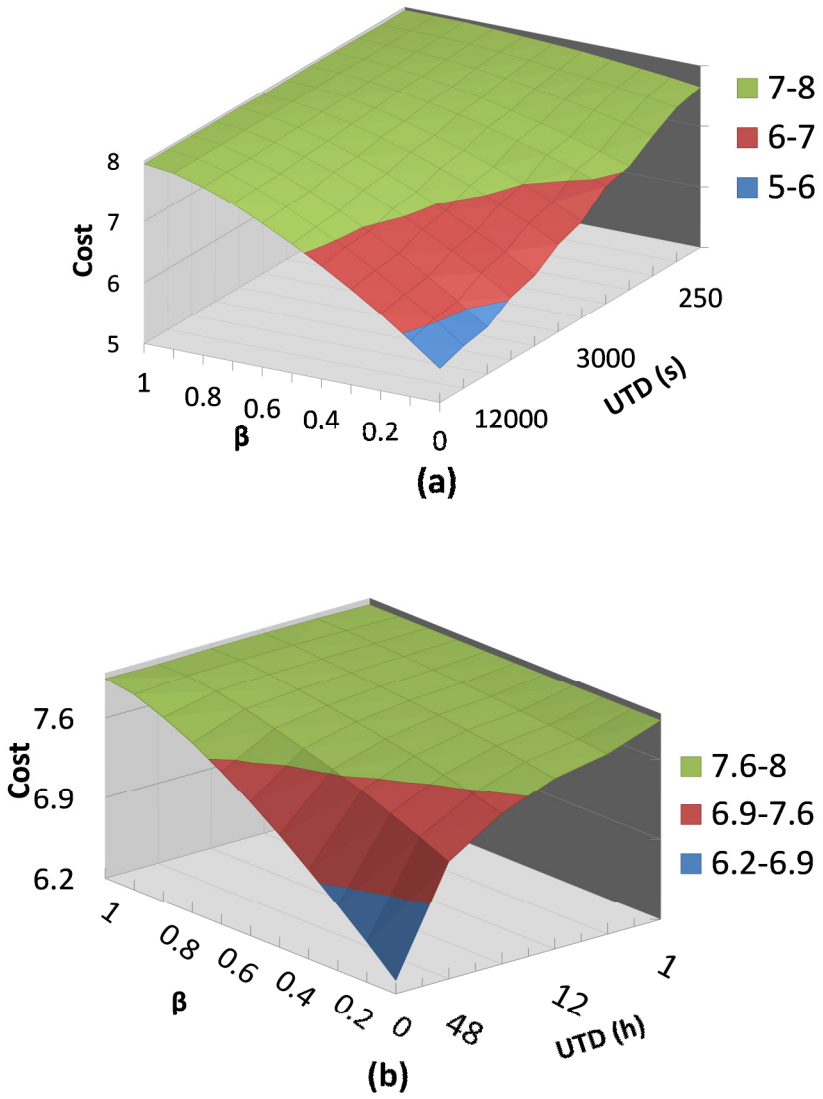


Figure 5-8: Object provisioning cost for the (a) Infocom 06 trace and (b) UCSD trace

Figure 5-8(a) reports the provisioning cost for a mobile network derived from the reduced Infocom 06 trace. The following observation can be made from this graph. First, with increasing β the provisioning cost also increases. This is expected as higher β implies higher rebate which, in turn, reduces the benefit of cooperative caching. The

second observation is that when users opt to wait longer (higher user tolerable delay), they have a higher chance to meet more number of nodes and therefore the overall probability of finding the requested object within the network increases. This in turn results in lower provisioning cost. The impacts of higher UTDs are more pronounced for small βs.

UCSD trace: Average provisioning cost from the UCSD trace is reported in Figure 5-8(b). Similar to the Infocom 06 scenario, as expected, the provisioning cost shows a decreasing trend when the UTD goes up, and the cost reduces with decreasing β. However, for the UCSD trace cost reduction starts at a higher UTD (i.e. 8 hours) compared to that for the Infocom trace (i.e. 10 minutes). The main reason for this is higher inter-contact times for the UCSD case in comparison to the Infocom case. An interesting observation in Figure 5-8(b) is that the minimum provisioning cost for the UCSD trace with 273 users is 6.3 (when β is *0* and UTD is *160* hours). This number for the Infocom trace with 98 users is about 5.5 (when β is 0 and UTD is 20000 seconds). This result shows that many of the users in the UCSD trace never meet each other (even when we increase the UTD to 160 hours). To confirm this we measured the percentage of isolated nodes for Infocom and UCSD trace and observed that at least 20 percent of nodes in UCSD trace are isolated. This number of Infocom trace was close to zero.

The second reason for higher provisioning cost in UCSD in spite of having more number of nodes in that network is that the connections between users in one single community is not as strong as connections in the Infocom 06 trace. To measure the strength of connections in a community we measure the community cohesion factor. The cohesion factor for the Infocom trace when UTD is at least 10000 seconds is always greater than

0.5, for UCSD however this number never goes beyond 0.35 even when we increase UTD to 168 hours.

5.7.3 Comparison with non-hierarchical Split Caching

So far, all results in Section 5.7 correspond to hierarchical split-caching as proposed in Section 5.4. In this section we compare those results with non-hierarchical split caching (i.e. single level split caching as presented in 3.8 Chapter 3). This single level split does not leverage information about community abstraction. Results are compared for HCMM mobility model and both Infocom 06 and UCSD mobility traces.

The minimum provisioning cost for single level split is numerically found by measuring the performance of Split caching over all possible values of the split factor λ. The percentage improvement of community based caching (i.e. with hierarchical split cache) with respect to the best result obtained by single level caching is reported in the following graphs.

Figure 5-9(a) reports the percentage improvement of hierarchical split caching on single level split caching for the HCMM model. Initially when UTD is small, community based hierarchical split-caching caching shows no improvement over the single level split caching. With small UTDs, the number of contacts between nodes is also very low which eliminates any chance of cooperation with in-community peers. In this case, the object provisioning cost can be reduced only by local caching and therefore performance of hierarchical based caching and single level split caching are the same. By increasing UTD however nodes have higher change to meet other in-community nodes. Intuitively the number of contacts among nodes in the same community is much higher than the number of contacts between nodes in different communities. Considering this fact in its

design, community based hierarchical caching is able to take advantage of cooperation between nodes in the same community. The single level split caching, however, lacks this feature, which leads to higher provisioning costs.

The percentage improvement generally reduces as we increase the rebate to download cost ratio (i.e. β) because by increasing β the cooperation loses its role in cost reduction. In the extreme case, when $\beta=1$, the effects of cooperation on provisioning cost completely disappears.

A very similar trend in Figure 5-9(b) can be observed for the Infocom 06 trace. Unlike for the HCMM case, however, the percentage improvement of hierarchical based caching in the Infocom 06 trace starts falling at relatively higher values of UTD. The reason is that by increasing UTD, the communities merge together. In the extreme case, when UTD is large enough, all nodes in the network will be in the same community. In this case, the performance of split caching and community based caching would be the same. For the Infocom '06 trace, only two communities are detected for large UTDs. By increasing UTD even more, one community starts growing and the other one shrinks. Note that performance improvement of hierarchical based caching is maximized when the sizes of all communities are the same. Performance improvement for the UCSD trace is shown in Figure 5-9(c). Similar to the two other traces, the community based caching shows higher improvement for lower βs and higher UTDs.

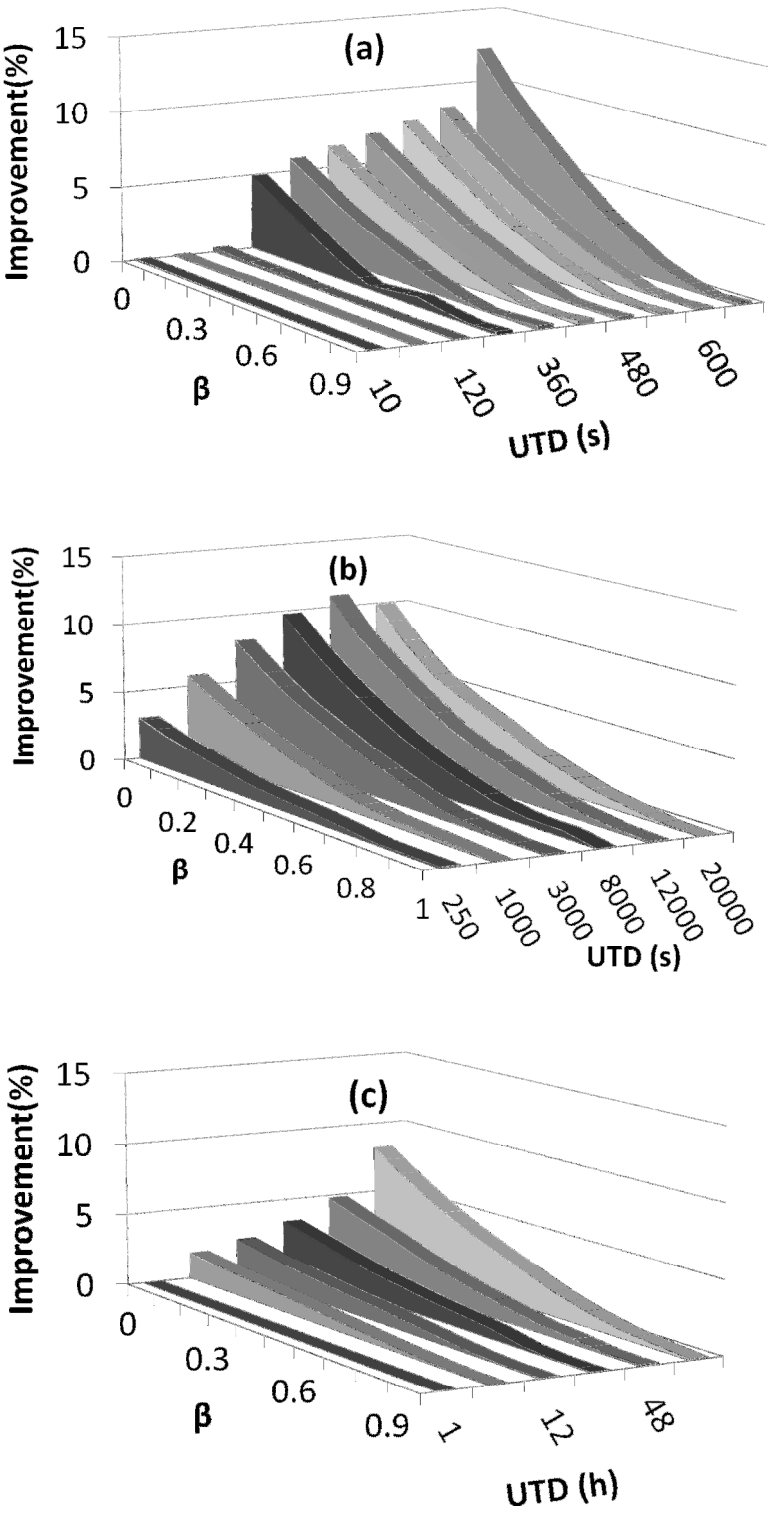


Figure 5-9: Performance improvement by using community based hierachical split cashing with respect to single level split for (a) HCMM model (b) Infocom '06 trace, and (c) UCSD trace

5.7.4 Performance of Community Detection Algorithms

In this section we compare the impacts of different community detection algorithms (see Section 5.4) on object provisioning cost. The UCSD trace with UTD of 48 hours has been used for these comparisons presented in Figure 5-10.

The object provisioning costs for *fastcommunity* and *fastfold* community detection algorithms are very similar and both of them provides better cost compared to the single level split caching. Both of these algorithms are heuristic in nature, and they work based on maximizing the network modularity as described in Section 5.4. The community detection algorithm *clique* provides lower cost solutions when compared to the other two. The *clique* algorithm is also more computationally expensive compared to the other two, which are heuristics based.

With *clique* all nodes in a community will have a strong connection and therefore the cohesion factor in community found by *clique* is very high which results in lower provisioning cost for clique as it is shown in Figure 5-10.

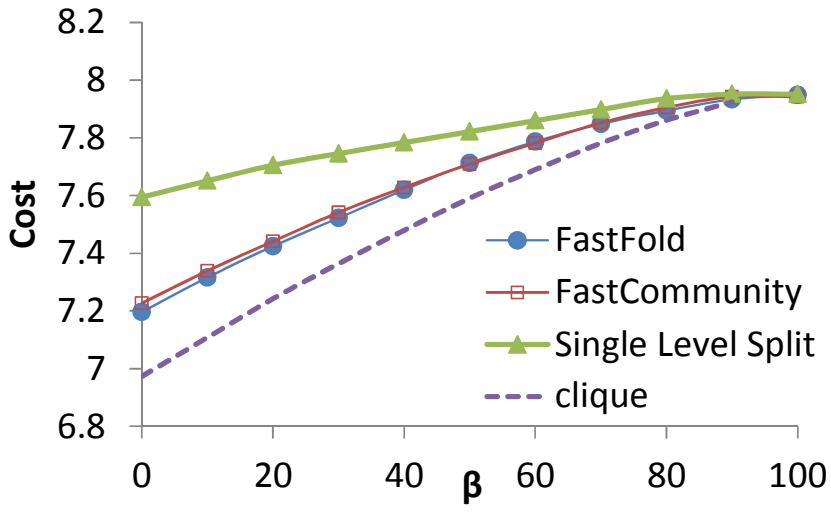


Figure 5-10: Cost of hierarchical and non- hierarchical split caching with different community detection algorithms

5.8 Summary and Conclusions

Drawing motivation from Amazon's Kindle electronic book delivery model, this chapter develops a set of practical network, service, and pricing models which are used for creating a cost- optimal cooperative caching strategy in Mobile Social Wireless Networks. It was shown that the proposed strategy can minimize content provisioning cost under static and a range of various user mobility scenarios with and without social community abstractions. In addition to proving its optimality, we construct analytical models and simulation results for evaluating the performance of the proposed strategy with prevalent mobility models and real human mobility traces from the literature.

Chapter 6 : IMPACTS OF USER-SELFISHNESS

6.1 Selfish Behavior

In 3.3 we introduced a pricing model in which the network usage cost and rebates are paid by the content provider. The scope for earning peer-to-peer rebate may promote selfish behavior in some users. A selfish user is a user that deviates from the network wide optimal caching policy to earn more rebates. Any deviation from the optimal policy is expected to incur higher network wide provisioning cost. In this chapter we analyze the impacts of such selfish user behavior on the object provisioning cost and on the earned rebate in a social wireless network (SWNET). It can be shown that beyond a selfish node population, the amount of per-node rebate for the selfish nodes is lower than that for the non-selfish nodes. In other words, when the selfish node population is beyond a critical point, the selfish behavior ceases to produce more benefit from a rebate standpoint.

There is a second pricing model in which the network usage cost and rebates are paid directly by end consumers. In this case the selfish nodes attempt to reduce their provisioning cost and maximize the earned rebate. We also analyze the impacts of such selfish behavior on the provisioning cost of selfish and non-selfish users within the *SWNET* settings.

6.2 User Selfishness and its Impacts

In Chapter 3 we computed the cost and rebate in a cooperative network where all nodes run the split replacement policy with the optimal λ. The impacts of selfishness are analyzed in this section. Selfishness in this context is defined as when one or multiple nodes execute split replacement with non-optimal λ values to earn more rebates (or to

reduce the provisioning cost in the second pricing model). Note that the selfish nodes still run cooperative caching but with a non-optimal λ.

Degree of selfishness in an *SWNET* is modeled by the parameters η, the number of selfish nodes and λ_S which is the non-optimal split-factor chosen by those nodes. The value of λ_S can be smaller or larger than λ_O, which is the optimal λ used by the non-selfish nodes. Average provisioning cost in a m-node network with both non-selfish and selfish nodes can be written as:

$$Cost = \frac{\eta Cost_{selfish} + (m-\eta) Cost_{non-selfish}}{m} \quad (6\text{-}1)$$

6.3 Cost and Rebate for Non-Selfish Nodes

6.3.1 Cost Computation

In order to compute $Cost_{non\text{-}selfish}$ using Eqn. 6-1, we need to compute the quantities P_L and P_V for the non-selfish nodes. The steady state cache status for both non-selfish and selfish nodes for the case $\lambda_S < \lambda_O$ is demonstrated in Figure 6-1.

Local hit rate: The non-selfish nodes store $\lambda_O C$ most popular objects in the duplicate segment of their cache (areas A1 and A2 in Figure 6-1), and fill the rest with unique objects (area A3 in Figure 6-1). Assuming unique objects are uniformly distributed in all nodes, the quantity P_L for a non-selfish node can be computed as:

$$P_L^C = H_D^C + \frac{1-\lambda_o}{\mu} H_U, \quad (6\text{-}2)$$

where $\mu C = \big((m-\eta)(1-\lambda_o)+\eta(1-\lambda_s)\big)C$ represents the total number of unique objects stored in an *SWNET* partition. The quantity $H_U = f(\mu C + \lambda_o C) - f(\lambda_o C)$ represents the corresponding hit rate for all unique objects in the partition. The first term $H_D^C = f(\lambda_o C)$ refers to the hit rate contributed by the duplicated objects and the second term refers to the hit rate contributed by the unique objects stored in a non-selfish node.

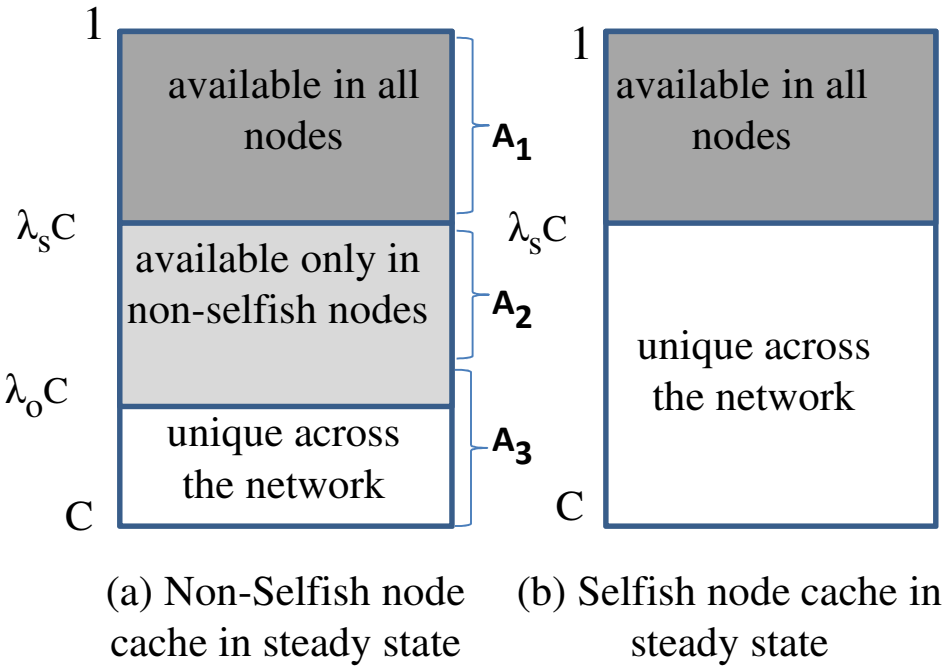


Figure 6-1: Cache status at steady state

Remote hit rate: After a local search fails in a node, it searches its local partition for the desired object. The desired object can only be found in the unique area of the remote caches in the partition since the duplicate area (which is same at all the nodes) has already been scanned during the local search. Therefore, the probability of finding the desired object in the partition (after its local search failed) is equal to the corresponding hit rate contributed by all the unique objects stored in the partition except those unique objects stored in the requesting device. Thus, the remote hit rate in a cooperative node is equal to:

$$P_V^C = \left(1 - \frac{(1-\lambda_0)}{\mu}\right) H_U \tag{6-3}$$

Substituting the value of P_L and P_V from Eqns. 6-2, 6-3 in Eqn. 3-3, the cost for provisioning objects to the non-selfish nodes can be simplified as:

$$Cost_{non-selfish} = \left(1 - \frac{((1-\beta)\mu+\beta(1-\lambda_o))C}{\mu C} H_U - H_D^C\right) C_d. \tag{6-4}$$

Note that H_U in the above equation must be computed as:

$$\begin{cases} H_U = f(\mu C + \lambda_o C) - f(\lambda_o C) \; when \; \lambda_s < \lambda_o \\ H_U = f(\mu C + \lambda_s C) - f(\lambda_s C) \; when \; \lambda_s \geq \lambda_o \end{cases} \tag{6-5}$$

6.3.2 Rebate Computation

The amount of rebate earned by a node depends on the number of requests generated in the network for objects stored in its local cache. In addition to the globally available objects (stored in the area A1 in Figure 6-1) all non-selfish nodes maintain certain duplicated objects in their cache which are not available in the selfish nodes (objects in area A2 in Figure 6-1). Thus, in addition to the unique objects (area A3 in Figure 6-1), each non-selfish node provides certain duplicated objects (stored in the area A2) to the selfish nodes. Therefore, amount of rebate per requested object in the network for non-selfish node can be written as:

$$Rebate_{non-selfish} = \left(\frac{1-\lambda_O}{\mu} H_U(m-1) + \frac{f(\lambda_o C) - f(\lambda_s C)}{m-\eta}\eta\right) \tag{6-6}$$

The first term in Eqn. 6-6 indicates the corresponding rebate for providing unique objects (from area A3) to all other nodes in the network, and the second term indicates the rebate for providing certain duplicated objects (from area A2) to the selfish nodes. The hit rate contributed by the duplicated objects that are not available in the selfish

nodes is equal to $f(\lambda_o C) - f(\lambda_s C)$. It is assumed that the generated requests from the selfish nodes are serviced by all non-selfish nodes in a uniform manner (that is why the quantity $f(\lambda_o C) - f(\lambda_s C)$ divided by m-η). Note when $\lambda_S > \lambda_O$, the selfish nodes maintain more duplicated objects, and therefore the second term in Eqn. 6-6 vanishes. The rebate in this case can be written as:

$$Rebate_{non-selfish} = \left(\frac{1-\lambda_O}{\mu} H_U(m-1)\right)\beta C_d. \quad (6\text{-}7)$$

6.4 Cost and Rebate for Selfish Nodes

6.4.1 Cost Computation

Similar to the non-selfish nodes, cost and rebate for the selfish nodes are computed using Eqn. 4 based on the local and remote hit rates. P_L for a selfish node can be computed as:

$$\mathrm{P_L^N} = \mathrm{H_D^N} + \frac{1\text{-}\lambda_o}{\mu}\mathrm{H_U}, \quad (6\text{-}8)$$

where $H_D^N = f(\lambda_s C)$ refers to the hit rate contributed by the duplicated objects stored in a selfish node, $\mathrm{H_U}$ is the hit rate contributed by all unique objects in the partition, Remote hit rate for a selfish node can be computed as:

$$\mathrm{P_V^N} = \left(1\text{-}\frac{(1\text{-}\lambda_S)}{\mu}\right)\mathrm{H_U}. \quad (6\text{-}9)$$

Substituting the value of P_L and P_V from Eqns. 6-8, 6-9 in Eqn. 3-3 the cost for selfish nodes can be simplified as:

$$\mathrm{Cost_{selfish}} = \left(1\text{-}\frac{(1\text{-}\beta)\mu+\beta(1\text{-}\lambda_S)}{\mu}\mathrm{H_U}\text{-}\mathrm{H_D^N}\right)\mathrm{C_d}. \quad (6\text{-}10)$$

6.4.2 Rebate Computation

Assuming that unique objects are uniformly distributed among all network nodes, the rebate for a selfish node when $\lambda_S < \lambda_O$ can be computed as:

$$Rebate_{selfish} = \left(\frac{1-\lambda_s}{\mu} H_U(m-1)\right)\beta C_d. \quad (6\text{-}11)$$

When $\lambda_S > \lambda_O$, the rebate for selfish nodes changes to:

$$Rebate_{selfish} = \left(\begin{array}{c}\frac{1-\lambda_s}{\mu} H_U(m-1) + \\ \frac{f(\lambda_s C)-f(\lambda_o C)}{\eta}(m-\eta)\end{array}\right)\beta C_d. \quad (6\text{-}12)$$

In the above equation, the first term indicates the amount of rebate a selfish node earns by providing its unique objects to all other nodes, and the second term represents the rebate earned by providing its duplicated objects to the non-selfish nodes. The quantity $f(\lambda_s C) - f(\lambda_o C)$ corresponds to the hit rate contributed by the objects that are not available in non-selfish nodes and duplicated across all selfish nodes. These objects are provided only to the $m - \eta$ non-selfish nodes.

6.5 Performance under First Pricing Model

In this section we analyze the impacts of user selfishness on the object provisioning cost when this cost is paid by content provider (i.e. similar to kindle model). In this case, earning higher rebate is the main motivation for selfish nodes for deviating from the optimal policy. We expect to see a higher provisioning cost under the presence of user selfishness.

6.5.1 Networks with Single Selfish Node

Figure 6-2 demonstrates the amount of rebate for each object request when there is exactly one selfish node in the network. With a single selfish node, choosing any λ_S that is different from the optimal λ increases the amount of rebate for the selfish node. The maximum value of earned rebate, however, depends on the value of λ_{opt} which is a function of β. For example, when β=0.9 (i.e. when the optimal value for λ is around 0.81), a selfish node can maximize its earned rebate by setting λ_S to 0. On the other hand, when β=0.5 (i.e. when λ_S is equal to 0.21) thr selfish node's rebate is maximized when its λ_S is set to 1. In summary, a single selfish node can maximize its own rebate by setting λ_S to either 0 or 1, whichever is farther than λ_{opt}. The "No Selfishness" marked points in Fig. 7(a,b) correspond to the situation where $\lambda_S = \lambda_{opt}$. A notable observation from Fig. 7(b) is that when the network usage cost is paid by the content provider (i.e when we use the first pricing model based on Kindle e-book delivery strategy), the main reason for deviating from optimal policy by selfish nodes is to earn higher rebates. The selfish policy depends on the value of β which determines the amount of rebate earned for each provided object.

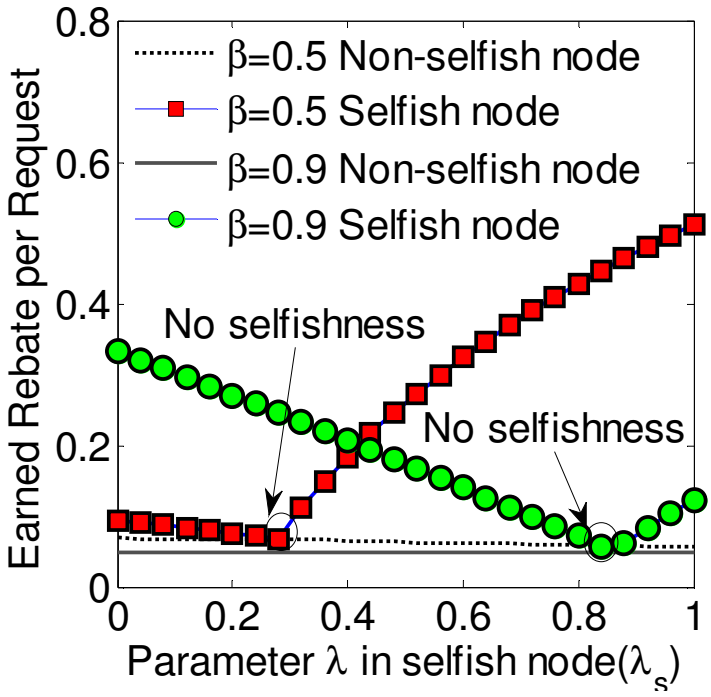


Figure 6-2: Rebate per request with one selfish node in the network

6.5.2 Networks with Multiple Selfish Nodes

6.5.2.1 Impacts on the provisioning cost

Figure 6-3(a) depicts the impacts of selfish node-count on the cost of provisioning objects when β=0.9. As expected, any deviation from the optimal policy increases the average provisioning cost in the network. For β=0.9, the selfish nodes choose λ_S=0 to maximize their rebates. Meaning, selfish nodes store only unique objects in their cache which in turn increase their provisioning cost. By increasing the number of selfish nodes, the number of uniquely stored objects in the network also increases. This new set of unique objects reduces the provisioning cost for the non-selfish nodes.

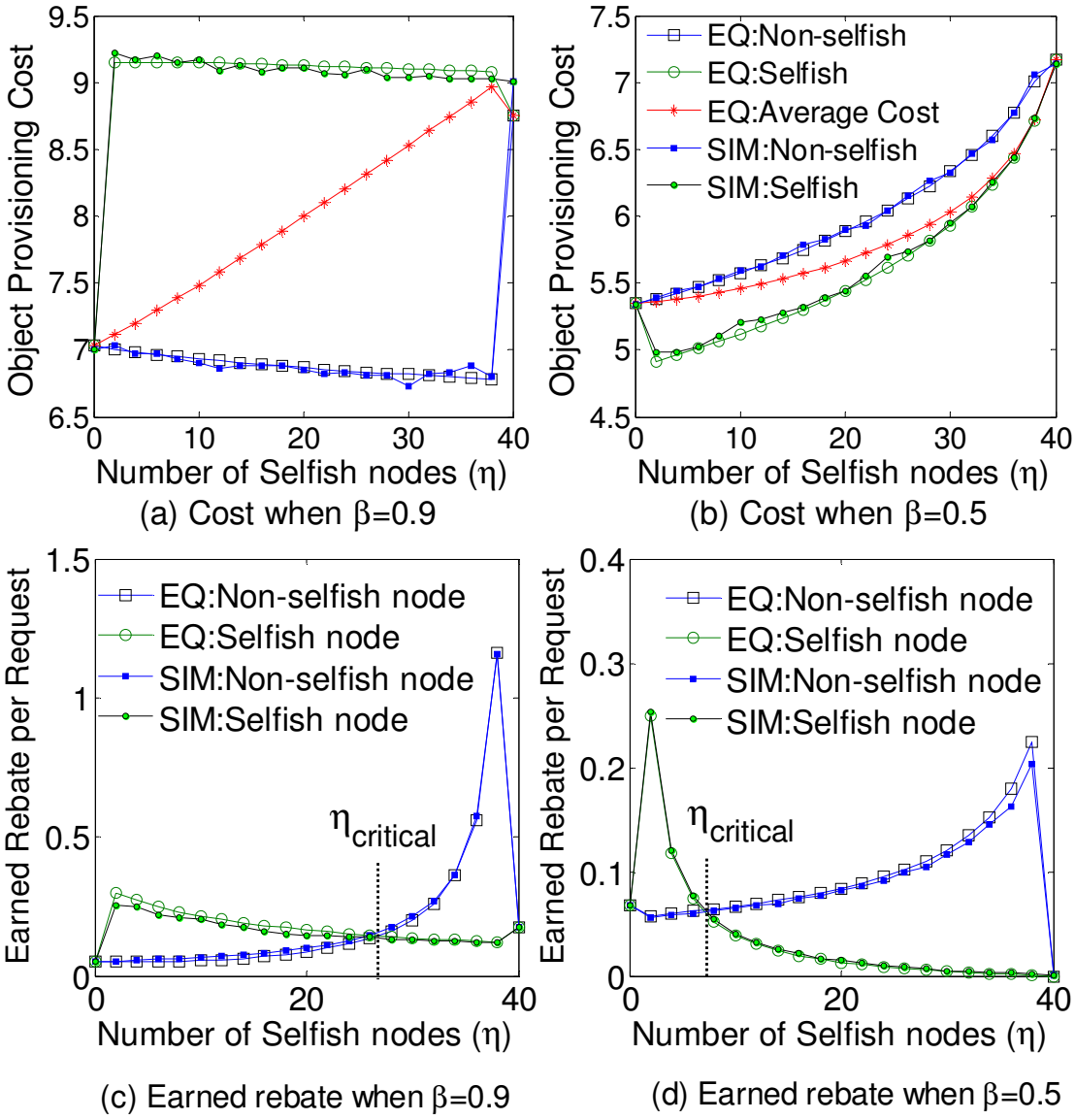


Figure 6-3: (a,b) Object provisioning cost and (c,d) Earned rebate per request for non-selfish and selfish nodes

Figure 6-3(b) demonstrates the cost of provisioning objects to the selfish nodes, the non-selfish nodes, and the network wide average when β is set to 0.5. In this case, the selfish nodes set λ_S to 1, causing them to store the most popular objects in their local cache. This helps the cost of object provisioning to the selfish nodes to come down. The cost for the non-selfish nodes, however, increases in the presence of selfish nodes due to the following two reasons: (1) a selfish node prevents other cooperative nodes to store

popular objects by storing the most popular objects in its cache (remember that a non-selfish node will not store a duplicated object in the unique area of its cache). (2) a selfish node wastes the global cache capacity by filling its cache with duplicated objects. As a result, less number of objects are stored in the network, which in turn reduces the chance of finding a requested object in remote caches. The excellent agreement between the analytical (labeled as EQ) and the simulation results (labeled as SIM) proves the correctness of Eqns. 6-2 to 6-12.

To earn the maximum amount of rebate, selfish nodes set the value of λ_S to either 0 or 1. It can be seen in Figure 6-3(a,b), depending on the chosen value of λ_S, the impacts of number of selfish nodes on provisioning cost can be substantially different.

Impacts on the Rebate

With the first pricing model, earning higher rebate is the only motivation for a node to run the selfish policy. From the rebate standpoint, a *steady state* can be defined as a situation in which a node cannot deviate from the optimal caching policy to earn higher rebate. In this section we demonstrate the existence of such steady state in a typical social wireless network.

Figure 6-3(c) represents the amount of rebate per request earned by the selfish and the non-selfish nodes when β is set to 0.9. Initially, when only few nodes deviate from the optimal policy, they are able to supply enough unique objects to the non-selfish nodes so that the earned rebate is higher. By increasing the number of selfish nodes, the rebate per request for each selfish node reduces for two reasons: 1) number of requests from non-selfish nodes becomes less, and 2) rebate must be shared among more number of selfish

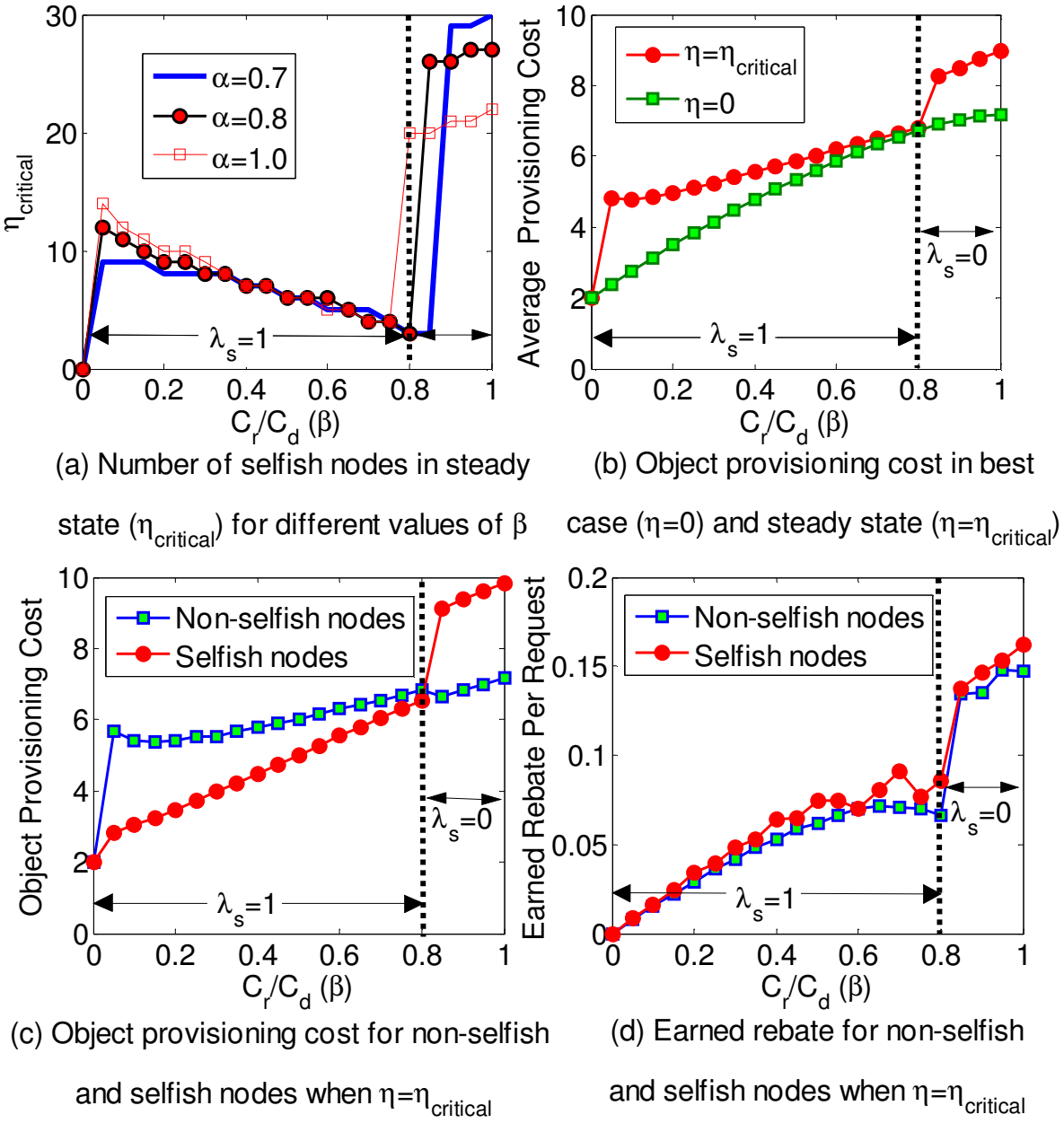
nodes. When the number of selfish nodes reaches a critical value $\eta_{critical}$, the rebate for selfish and non-selfish nodes become equal. In fact, when a node chooses a selfish policy while there are $\eta_{critical}$ selfish nodes in the network, its rebate become less than that of the non-selfish nodes. This leads to an important claim, namely, having more than $\eta_{critical}$ selfish nodes in the network does not serve any purpose for the selfish nodes.

Figure 6-3(d) represents the amount of rebate for selfish and non-selfish nodes when β is set to 0.5. The value of λ_s at the selfish nodes is set to 1, so that the earned rebates by those nodes are maximized. Observe that for higher β, the value of η is also higher. Meaning, more number of nodes can execute the selfish policy and still receive higher rebates compared to the non-selfish nodes.

6.5.3 Steady State Analysis

This section presents analysis in the steady state, which is when a network contains exactly $\eta_{critical}$ number of selfish nodes. Figure 6-4(a) depicts the impacts of β on $\eta_{critical}$, which represents the maximum number of nodes that can run the selfish policy and still get higher rebates compared to the non-selfish nodes. When β is small, the optimal λ_{opt} is also small which means each non-selfish node stores only a very few popular duplicated objects in its cache. Therefore, a selfish node can earn a high rebate by storing the most popular objects in its local cache (i.e. choosing λ_s=1) and serving a large number of requests from other nodes. With increasing β, the quantity λ_{opt} increases,

which causes each node to store more popular objects in its local cache and therefore, the number of remote requests to the selfish nodes reduces.



(a) Number of selfish nodes in steady state ($\eta_{critical}$) for different values of β

(b) Object provisioning cost in best case (η=0) and steady state (η=$\eta_{critical}$)

(c) Object provisioning cost for non-selfish and selfish nodes when η=$\eta_{critical}$

(d) Earned rebate for non-selfish and selfish nodes when η=$\eta_{critical}$

Figure 6-4: Analysis of rebate and object provisioning cost in steady state (i.e. η=$\eta_{critical}$)

This in turn reduces the rebate difference between the selfish and the non-selfish nodes, and therefore less number of nodes can run the selfish policy. This explains the decreasing trend (see Figure 6-4(a)) of $\eta_{critical}$ as β changes from 0 to 0.8. At β=0.8, the amount of rebate for the selfish nodes is very close to the amount of rebate for the non-selfish nodes. As a result, only very few nodes can benefit from running the selfish policy. For $\beta > 0.8$, the selfish nodes must set λ_s to 0 in order to get higher rebates compared to the cooperative nodes. In this case, the difference between the rebate earned by the selfish and the non-selfish nodes becomes very high which encourages a lot of nodes to choose selfish policy, thus drastically increasing the quantity $\eta_{critical}$.

Figure 6-4(b) demonstrates the average object provisioning cost in the presence of $\eta_{critical}$ selfish nodes. For β=0, as there is no selfish node in the network, the provisioning cost in this case represents the minimum possible value. In other cases, as expected the average provisioning cost is always higher than the provisioning cost when all nodes run the optimal policy.

Observe that the impacts of selfishness for small βs are always higher than those for large βs. For higher β, the non-selfish nodes store more popular objects in their local cache and they become less sensitive to the presence of selfish nodes. For β=0.8, the average provisioning cost in the presence of selfish nodes is very close to the minimum possible provisioning cost. For β greater than 0.8, the selfish nodes set λ_s to zero and as demonstrated in Figure 6-4(a), the number of selfish nodes $\eta_{critical}$ becomes very large.

Because of too many selfish nodes, the difference between the cost of provisioning objects to the non-selfish and selfish nodes becomes noticeable again.

Figure 6-4(c) depicts the cost of object provisioning for the selfish and the non-selfish nodes in the presence of $\eta_{critical}$ selfish nodes in the network. For small βs, the cost of provisioning objects to the selfish nodes is much lower than that for the non-selfish nodes. This is because for small βs the selfish nodes set λ_s=1 and store popular objects locally. Therefore, a high percentage of requests in selfish nodes are satisfied locally without any provisioning cost. The difference between non-selfish and selfish nodes becomes less as β increases because due to higher λ_{opt}, the non-selfish nodes also start storing more popular objects. After β=0.8, the cost for provisioning to the selfish nodes becomes higher than that to the non-selfish nodes. The reason is by choosing λ_s=0, a selfish node is deprived of having popular objects and only a few percentage of requests in selfish nodes in this case can be satisfied from the local caches, which in turn increases the provisioning cost.

Figure 6-4(d) depicts the amount of rebate earned by the selfish and the non-selfish nodes. Observe that the amount of rebate earned by the selfish nodes is close and always higher than the amount of rebate earned by the non-selfish nodes. Adding even a single selfish node beyond $\eta_{critical}$ brings the amount of rebate for the selfish nodes below that of the non-selfish nodes. The sharp jump in the rebate at β=0.8 is because of switching from λ_s=1 to λ_s=0 .

6.5.4 Impacts of rebate on Node Participation

Up to this point, all the presented analysis and performance results are based on the assumption that the end-consumers participate in cooperative caching regardless of the value of the rebate C_r. They can be selfish (i.e. when use λ_S) or non-selfish (i.e. using λ_{opt}), but they do cooperate. In a more realistic situation, however, the participation of a node in cooperative caching is expected to be an increasing function of C_r, the amount of rebate offered for each peer-provided object. In other words, for larger C_r values, more number of nodes are expected to participate in collaborative caching. In this section we study the performance of caching under the first pricing model (i.e when provisioning cost and rebate is paid by content provider) and with three different increasing functions: f^1,f^2 and f^3 as defined below:

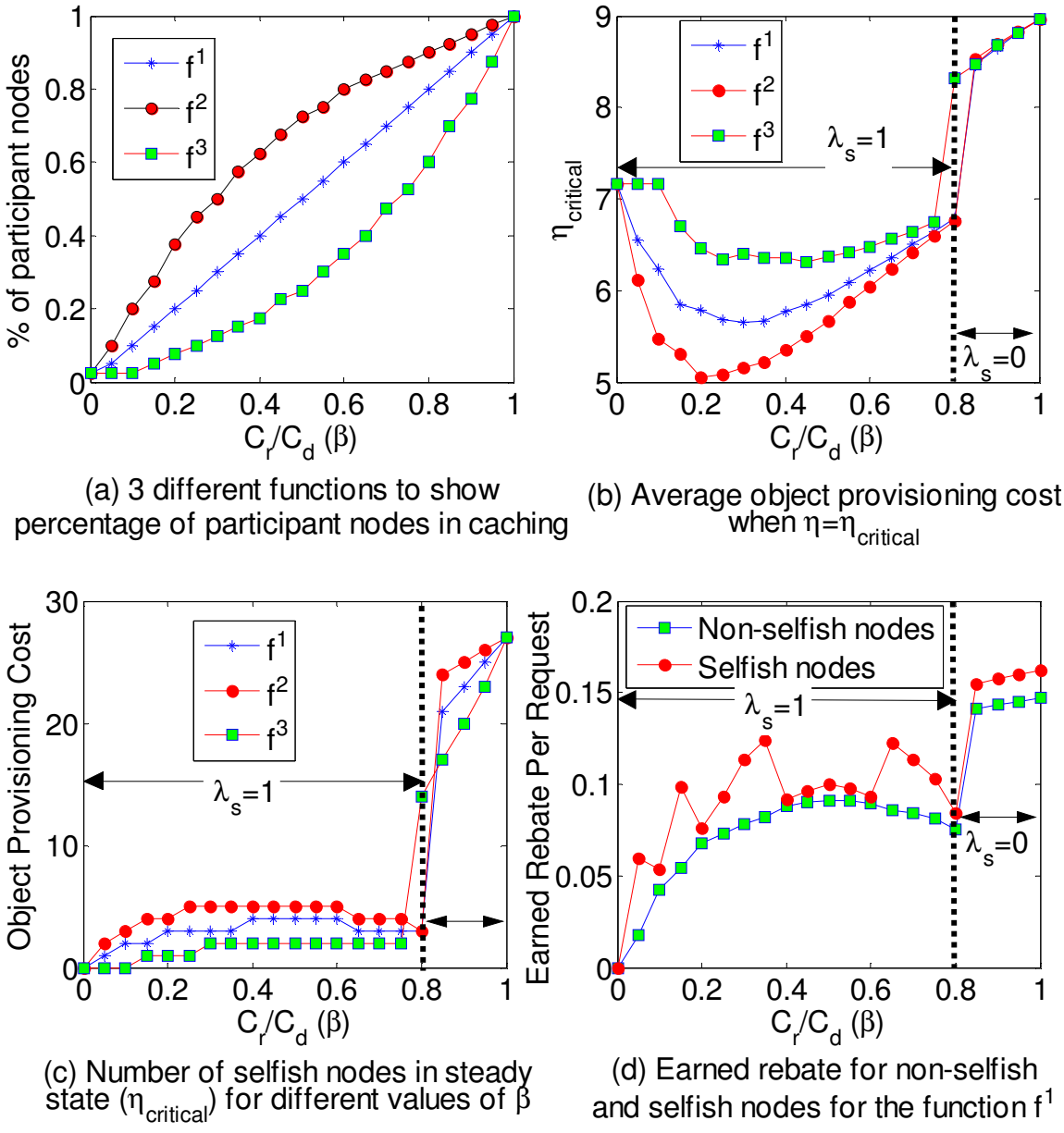
$$f^1: \phi = \beta$$

$$f^2: \phi = \left(1 - e^{-2\beta}\right)/(1 - e^{-2})$$

$$f^3: \phi = \left(1 - e^{2\beta}\right)/(1 - e^{2})$$

In the above functions, φ indicates the probability that a node participates in collaborative caching. With probability (1-φ), a node does not participate in caching; meaning all object requests by that node are served by direct downloads from the content provider's server. Three types of node can be distinguished in the network: 1) non-participating, 2) participating and non-selfish, and 3) participating and selfish.

Figure 6-5(a) depicts the variation of node participation as a function of the rebate-to-download-cost ratio β following the above three models. Figure 6-5(b) reports the network wide average object provisioning cost for the participating nodes when the system is in steady state, meaning $\eta=\eta_{critical}$. It can be observed that by increasing β, the object provisioning cost for the participant nodes initially reduces because more number of nodes participates in caching. However, for higher β, the impacts of collaborative caching reduce and the cost increases again. Also, the provisioning cost for f^2 is the least because the rate of increase of the number of participant nodes with β is highest in this function. Figure 6-5(c) depicts the steady state (see Section 6) selfish node population $\eta_{critical}$ under different β and participation dependency functions f^1 through f^3. The trend of $\eta_{critical}$ in Figure 6-5(c) is different than Figure 6-4(a) because of number of participant nodes in this case is much less. Figure 6-5(d) represents the network wide average amount of earned rebate for the participating selfish and non-selfish nodes for the linear function for f^1.



(a) 3 different functions to show percentage of participant nodes in caching

(b) Average object provisioning cost when $\eta=\eta_{critical}$

(c) Number of selfish nodes in steady state ($\eta_{critical}$) for different values of β

(d) Earned rebate for non-selfish and selfish nodes for the function f^1

Figure 6-5: Impact of β (rebate-to-download-cost ratio) on the number of participant nodes and cost and rebate

6.6 Performance under the Second Pricing Model

6.6.1 Networks with Single Selfish Node

In this section we analyze the impacts of user selfishness on provisioning cost when the cost is paid by the end users. In this case, the motivations for user selfish includes: a) minimizing the provisioning cost, and b) maximizing the earned rebate. Provisioning costs for both selfish and non-selfish nodes in the presence of a single selfish node are shown in Figure 6-6. It can be observed that for all β, the selfish nodes can run the split caching policy with parameter λ_S=1 to minimize their own provisioning costs.

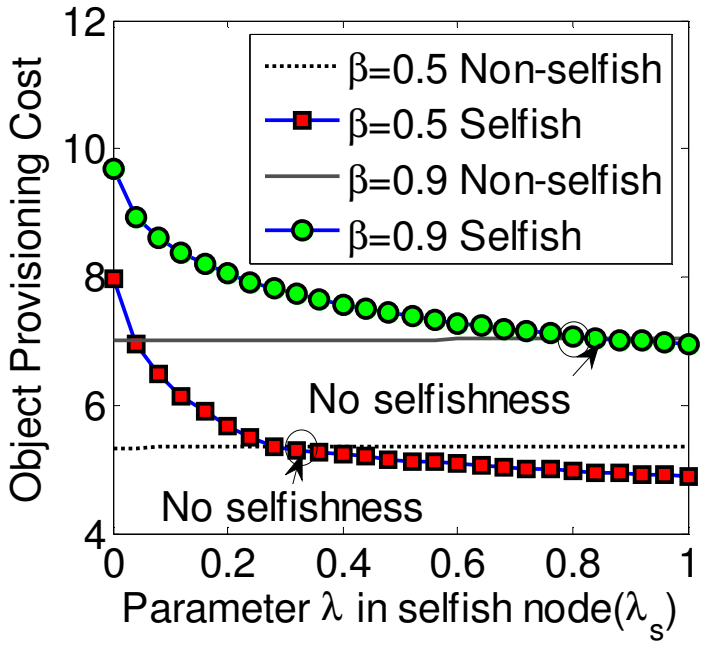


Figure 6-6: Object provisioning cost when only one selfish node exists in the network

6.6.2 Networks with Multiple Selfish Nodes

To analyze the impacts of selfish node population on provisioning cost two scenarios are studied. In the first scenario, we assume that the selfish nodes are not aware of the existence of other selfish nodes, Meaning, all selfish nodes set λ_S=1 to minimize

their own costs. In the second scenario we assume that such nodes collude and they choose a selfish policy according to the number of participating (and colluding) selfish nodes.

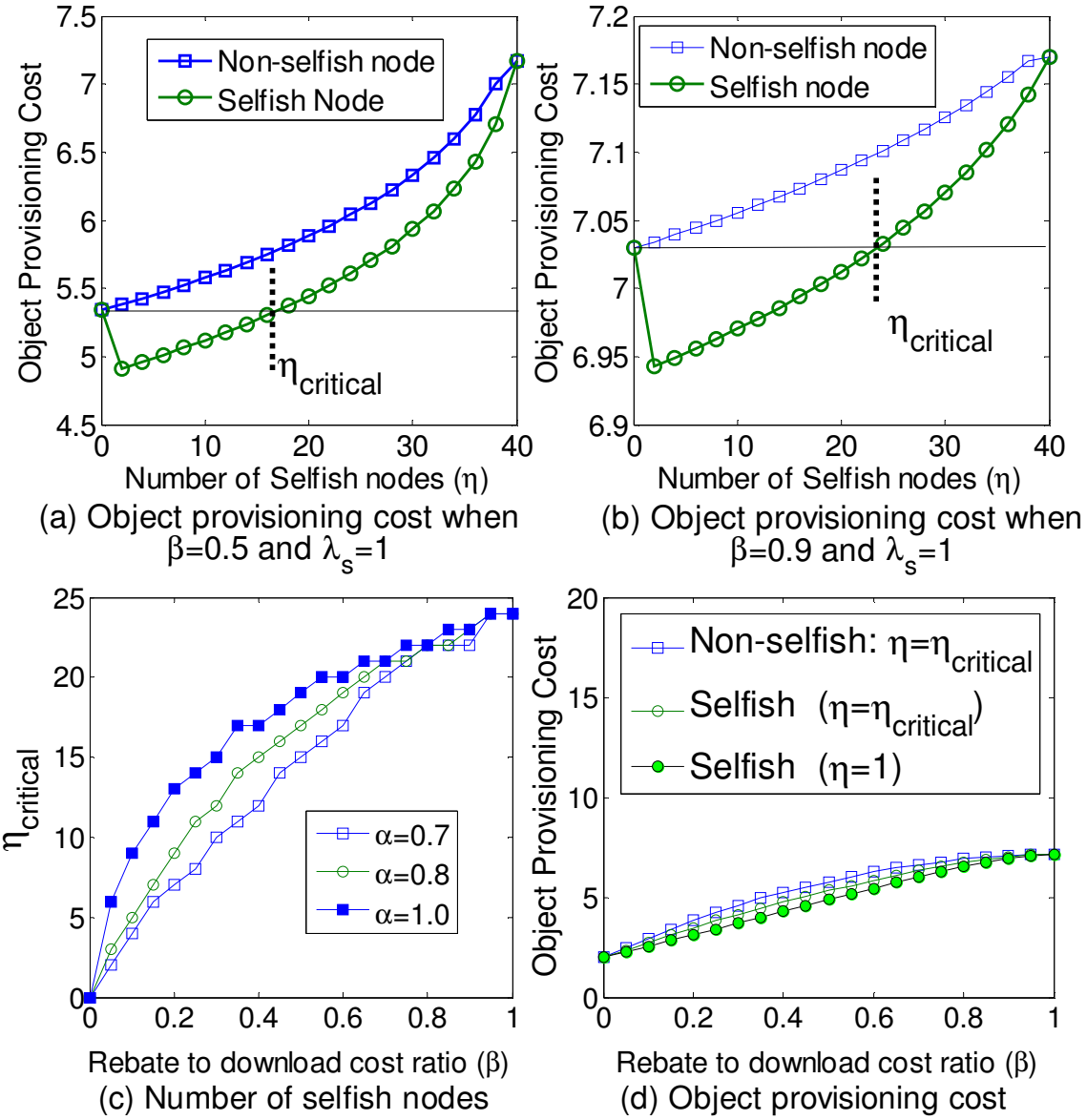


Figure 6-7: Impacts of selfishness without collusion between selfish nodes

6.6.2.1 Without Collusion

When a selfish node is not aware of other such nodes, it should choose a selfish policy that is likely to minimize its own cost. That policy is to set λ_S to 1. However, as

the number of selfish nodes increases, the resulting cost for the selfish nodes in fact may get worse than their cost when they run the optimal policy. Figure 6-7(a) and (b) demonstrate how the cost of provisioning for both selfish and non-selfish nodes changes when the rebate-to-download-cost ratio (β) is set at 0.5 and 0.9. As the number of selfish nodes (running caching policy with parameter λ_S =1) increases, the cost for both selfish and non-selfish nodes increases. Initially, the cost for selfish nodes is lower that that of non-selfish nodes. However, by increasing the number of selfish nodes the cost paid by the selfish nodes also increases. In fact, when the number of selfish nodes is beyond a critical η, the cost paid by the selfish nodes becomes larger than that when they run the non-selfish optimal policy. From this it can be concluded that in the presence of more than $\eta_{critical}$ selfish nodes in the network the motivation of user selfishness disappears when end users are responsible for paying the object provisioning costs.

Figure 6-7(c) demonstrates $\eta_{critical}$ with varying β (rebate-to-download cost ratio) between 0 and 1. Intuitively, as β increases, more nodes can afford to be selfish while gaining in terms of the provisioning cost. The values of $\eta_{critical}$ are shown for three different values of α (Zipf parameter). With higher α, the number of nodes that can run selfish policy while being able to reduce their cost of provisioning is also higher. The reason is with higher α, the percentage of requests for popular objects is higher and therefore, selfish nodes can reduce their cost of object provisioning by finding most of the requested objects in their local cache; thus leading to zero provisioning cost. As Figure 6-7(d) demonstrates, the cost for non-selfish nodes is always higher than the

optimal provisioning cost, and the cost for the selfish nodes is always lower than that for the non-selfish nodes. In steady state when there are $\eta_{critical}$ selfish nodes in the network, the cost for the selfish nodes is very close (but less than) to the optimal provisioning cost (i.e. when there is no user selfishness). When $\eta_{critical}$ is equal to 1, the cost of provisioning for the selfish nodes is less than the optimal cost.

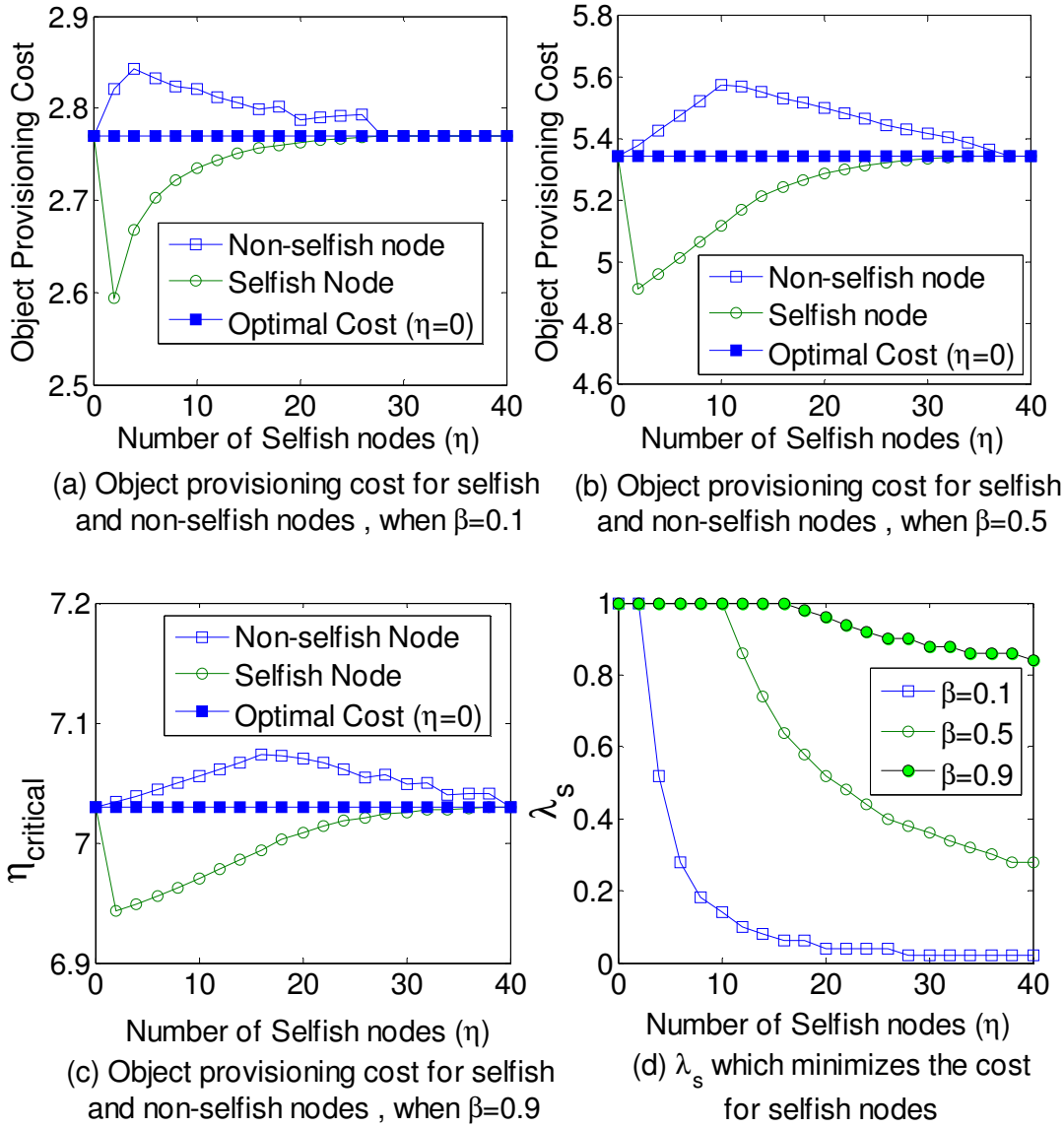


Figure 6-8: Impacts of selfishness with collusion between selfish nodes

6.6.2.2 With Collusion

The behavior of mutually aware and colluding user selfishness is analyzed in this section. It is shown that a selfish node is always able to keep its provisioning cost below the optimal cost when it is aware of the number of other selfish nodes in the network. In Figure 6-8(a, b and c) the provisioning cost for selfish and non-selfish nodes are demonstrated when β is equal to 0.1, 0.5 and 0.9 respectively. It can be observed that the selfish nodes can always maintain a lower than optimal cost by appropriately choosing λ_S. The provisioning cost for non-selfish nodes, however, is always higher than the optimal cost. One interesting observation from these graphs is that when all nodes are selfish, the selfish policy converges to the optimal policy. However, when only few nodes run selfish policy, the selfish and optimal policies are different which results in different costs of object provisioning for selfish and non-selfish nodes. Figure 6-8(d) demonstrates the desirable λ_S for the selfish nodes as a function of η, which is the selfish node population. It can be seen that the selfish nodes need to change their policy (i.e. to choose different λ_S) based on the number of other such nodes. This is because the knowledge of η helps keeping their cost always lower than the optimal cost (i.e. provisioning cost when there is no user selfishness). When the number of selfish nodes is very low, setting λ_S to 1 minimizes the cost for the selfish nodes. However by increasing the number of selfish nodes, λ_S must be set to lower values in order to maintain a lower that optimal cost.

6.7 Summary and Conclusion

In this chapter we analyzed the impacts of selfishness on performance of cooperative caching in social wireless networks. Any deviation from optimal split caching will increase the average provisioning cost in the network. We showed and proved both analytically and experimentally that selfish policy will not help a node to increase its rebate when number of selfish users is beyond a specific threshold.

Chapter 7 : COOPERATIVE CACHING FOR IMPROVING AVAILABILITY

7.1 Motivation

Wireless handheld devices and networked data applications on such devices have experienced unprecedented consumer penetration in recent years. The Apple iPod Touch, Nintendo DS, Sony PSP game console, and numerous other handheld platforms belong to this category of devices. An emerging list of data applications for such devices includes networked and peer-to-peer games, electronic books, newspaper and magazine readers, and thousands of Applications (*Apps*) that are written for devices such as iPod Touch.

Due to human mobility and the lack of complete coverage by WiFi access points, a *MSWNET* can be susceptible to intermittent disconnections to the Internet. This can result in partitions of devices that can communicate with each other using ad hoc routing protocols, but do not have internet connectivity. This lack of connectivity affects object (content) availability within *MSWNET* partitions for server-based applications such as electronic book and Apps downloads.

As an example, consider a game device (e.g. Nintendo DS) that downloads games directly from the game server. This device is not able to download the game while it is temporarily situated within a disconnected (from internet) *MSWNET* partition. However, with *cooperative caching* enabled, a DS may be able to download the game by searching other DSs and compatible devices within its local *MSWNET* partition.

High partition level availability ensures popular objects will be available within

MSWNET partitions when it disconnected from the Internet, and high node level availability ensures popular objects are available to individual nodes even when they are completely isolated from the rest of network and Internet gateway. Note that high node level availability does not necessarily lead to high partition level availability. For example, storing the same set of popular objects can maximize the average node level availability, however keeping multiple copies of the same object in the partition results in limited partition availability. The objective of this chapter is to design a *cooperative object caching* mechanism such that the availability of the popular objects can be maximized at both partition and node level.

7.2 Limitations of prior work

The common objective of the existing cooperative caching mechanisms for wireless networks is to achieve high object availability at the partition level. This is achieved by avoiding the storage of duplicated objects within a network partition. While improving partition level availability, these approaches offer low availability at the node level, because only one copy of each popular object exists within a partition. High node level availability is desirable so that the popular objects are available to nodes even when they are completely isolated without being connected to any other node. A limitation of the existing mechanisms is their inability to provide high availability at both partition and node levels. Related work is formally presented in Chapter 2.

7.3 Our approach and contribution

The main objective is to develop *cooperative caching* mechanisms that can improve object availability within *MSWNET* partitions as well as at individual nodes.

This is achieved by letting each node in a partition to store a set of objects while allowing certain level of duplication in the partition. To control the level of duplication, the cache space in a node is divided into two separate areas. The first area is dedicated for storing the most popular (duplicated across all nodes in the partition) objects to guarantee high availability within a completely isolated node. And the second area is dedicated to store partition-wide unique objects to achieve high level of availability at the partition level. Using a stochastic model we determine the boundary between the above partitions such that a required balance between node and partition level availabilities can be achieved. The contribution of the proposed cache partitioning mechanism is to provide high levels of node and partition availabilities, and at the same time to reduce the generated network traffic.

7.4 **Design Objectives**

The user perceivable performance indexes are: 1) *Partition Object Availability* (POA), which indicates the probability of an arbitrary object to be found within a partition; and 2) *Node Object Availability* (NOA), which indicates the probability of an object to be found within a node's local cache. NOA is a measure of availability for completely isolated nodes.

The network related performance index is the Generated Network Traffic (GNT) for fetching each object which is function of average hop-distance between a node and the object provider. In addition to the capacity load, GNT indicates the energy overhead of object access. The objective is to minimize GNT. A combined objective is to minimize the GNT while maintaining pre-specified levels of POA and/or NOA. It will be demonstrated that the proposed cooperative caching policy in this chapter is particularly

efficient in terms of meeting such combined objectives.

7.5 Cache resolution

Cache resolution addresses how to resolve an object request either locally or from the local partition. After a request is originated by an end-consumer (i.e. an application on a mobile device), it first performs a *local search* within its local cache. If it fails, the end-consumer (EC) performs a *network search* for the object within its local partition. If this step also fails, and the device is situated within a connected partition, the object is downloaded from the CP's server via a gateway mobile node, Access Point, and the Internet. If the partition is not connected, the object is deemed unavailable.

For searching an object within the local partition, a flooding based search mechanism is usually used. A Time to Live (TTL) based ring-search can be employed for constraining the scope of resolution. This can limit: a) the network costs, and b) the degree of caching cooperation [28]. The radius of this ring search (i.e. the search TTL) is termed as the *Domain of Cooperation* (DOC) because a node's cooperative behavior depends on the state of the caches within that ring. In other words, a node effectively cooperates with all nodes with hop-distance less than or equal to the operating DOC. When a node receives a search request for one of its locally cached objects, it sends a unicast ACK to the requester. Then the requester starts downloading the object from the responding node.

7.6 Cache management

Cache management refers to policies that control which set of objects are kept in the cache and how they are distributed in the partition. Cache management comprises

object prioritization and replacement policies.

DOC based object prioritization: A node adjusts its cooperative behavior depending on the state of other nodes' caches within a pre-specified DOC, so that the objectives POA, NOA, and GNT are met either individually or in a combined manner. A simple cooperation policy for a node is not to store an object if it has already been stored within the DOC. By doing this, the total number of different objects stored within the DOC can be increased. This in turn can increase the partition object availability. We refer to this model as *Prioritized Cooperation*.

With *Prioritized Cooperation* turned on, DOC acts as a tunable parameter that can control the degree of object duplication within a partition. The value of DOC can range from zero (i.e. no cooperation) to the diameter of the network (i.e. the maximum cooperation).

Object replacement: When a node fetches a new object, it executes a replacement policy in order to decide as to which object from its cache should be replaced, if any. Possible replacement policies include Random (RND), Least Recently Used (LRU), and Least Popular Object (LPO). LPO assumes that a device is aware of the server-tagged popularity of a downloaded object.

7.7 **Cooperative Caching (COOP)**

The protocol COOP in [28] develops a cooperative caching mechanism that relies on DOC based resolution with *Prioritized Cooperation*, and LRU based replacement in mobile networks. To our knowledge, COOP represents the most comprehensive cooperative caching approach in the literature. This is why we have chosen COOP to compare our proposed mechanism with. The key components of COOP are as follows.

7.7.1 Cache resolution

COOP uses both *local* and DOC based *network* search as detailed in Section 7.5. Additionally, it implements an optimization for reducing repetitive resolution costs by recording the address (called *profile*) of the node from which an object has been downloaded most recently. For subsequent resolutions for the same object by the same node, the *profile* node is first searched using unicast before the entire DOC is searched using flooding. This *profile* of historical access can often eliminate the overhead of flooding. In addition, COOP uses a *hop-by-hop* resolution in which when *local*, *profile*, and *network* searches all fail, a request is sent to the CP's server. Each intermediate node on the path between requester and the server investigates the request, and supplies the object to the requester if it finds the object in its local cache.

7.7.2 Cache management

COOP classifies each object stored within a node's cache as either primary or secondary. A primary object is unique within the pre-specified DOC whereas a secondary object is a duplicate. The protocol prioritizes unique object over duplicate objects during cache replacement. To implement this, it evicts a secondary object when there is no space left in the cache for accommodating a new downloaded object. When no secondary object exists in the cache, COOP uses a regular LRU replacement policy among all stored objects to choose a candidate from the cache to evict.

7.7.3 New flavors of COOP

COOP, as described above, corresponds to the baseline version reported in [28]. We have implemented that baseline and a number of new flavors of COOP by altering its

replacement policy and turning prioritization on and off. To distinguish among these different flavors, we use the following notation. COOP-P refers to the baseline implementation with prioritized cooperation, and COOP-N refers to a version with prioritization turned off. For both –P and –N, we implement RNDM, LRU and LPO replacement policies. For example, COOP-P-RNDM represents the prioritized COOP with RNDM replacement. It is found that for different topological characteristics of an *MSWNET* partition, these new flavors of COOP are able to generate better results compared to its baseline version.

7.8 Proposed Cooperative Split Caching (CSC)

7.8.1 CSC overview

Together with DOC, the proposed CSC strategy implements an additional mechanism for in-domain object duplication control. Unlike in COOP with only DOC based duplication control, by manipulating two duplication control parameters in CSC, it is possible to provide both high NOA and high POA at the same time, which is not feasible using COOP for reasons as described below.

Although there are certain differences in effective object duplications between COOP-P and COOP-N, the primary mode of duplication control in COOP is by varying the Domain of Cooperation during the cache resolution. Since the level of object duplication determines the interdependency between POA, NOA, and GNT, a finer control on object duplication is expected to provide better performance in terms of the combined objectives.

As DOC increases, due to reducing duplications, the NOA drastically reduces. This is because in order to store a large number of unique objects within a partition, few nodes

are forced to store less popular objects, thus leading to low NOA for those nodes. However, larger DOCs, due to lower duplication, can provide higher POA, since more unique objects are able to be stored within a partition. This indicates that the NOA and POA are conflicting requirements. COOP, equipped with only one duplication control parameter is not able to provide both high NOA and high POA simultaneously.

With the proposed CSC mechanism, however, by manipulating two duplication control parameters, it is possible to reconcile between the objectives NOA and POA. This is the primary contribution of CSC.

7.8.2 Cache Splitting

In Cooperative Split Caching (CSC), the cache space in each mobile device is divided into a *duplicate* area and a *unique* area. As shown in Figure 7-1, the *duplicate* area occupies $\lambda\ (0 \leq\ \lambda \leq 1)$ fraction of the available cache space and stores $\lambda.C$ most popular objects which are duplicated within the domain of cooperation. The unique area stores $(1-\lambda)C$ objects which are unique across the nodes within the domain of cooperation. The parameter λ is the second tunable quantity for duplication control in CSC. While DOC controls partition wide duplication by tuning the depth of ring search during cache resolution, λ controls the level of duplication within the domain of cooperation by shifting the emphasis between storing duplicated and unique objects.

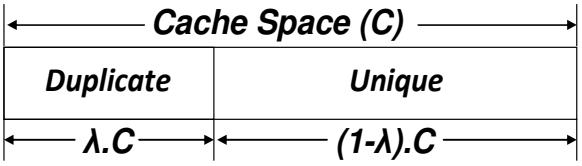


Figure 7-1: Cache partitioning in the *CSC* policy

The intuition behind this split is to let a limited number of popular objects to be

duplicated at all nodes within the cooperation domain. This duplication helps maintaining high *Node Object Availability* (NOA). Simultaneously, the unique part of the cache space can maintain sufficient number of unique objects so that the Partition Object Availability (POA) can be also maintained high. This is how, unlike in COOP, CSC can achieve both high NOA and POA by tuning the duplication control parameters DOC and λ to appropriate values.

7.8.3 Cache resolution and replacement

Cache resolution in CSC is based primarily on local search and DOC based network search as described in Section 7.5. Unlike in COOP, no profile based and hop-by-hop resolutions are performed. In the presence of mobility it is possible for few objects, which are meant to be domain-wide unique, to get duplicated due to the entry of new nodes within a domain. CSC implements a simple mechanism to fix such undesired duplications in the following manner. When a node sends a broadcast query for an object, it may receive multiple ACKs from different nodes within the domain of cooperation (i.e. the radius of ring search). Subsequently, the requester notifies all but only one ACK senders about the duplication. Upon receiving such notification, a node deletes the specific object from its cache only if it has been stored in the unique part of the cache.

Object replacement is based on popularity and the adopted cache split. When a new object is fetched from the CP's server, the downloading node attempts to replace a less popular object from its local cache (including both the duplicated part and the unique part). If no less popular object is found then the object is not stored in the cache. When a new object is fetched from the cooperation domain, the downloading node attempts to replace a less popular object only from the duplicated area of its local cache. If no less

popular object is found within the duplicate area then the object is not stored. In other words, objects fetched from the *local partition* are never cached within the unique part of the cache. Using the above logic, at steady state, all devices' caches in a cooperation domain will have the same set of objects in their *duplicate* areas.

Note that CSC with λ set to zero, is same as COOP-P-LPO. When λ is equal to zero, no space in cache is reserved for duplicate objects. In other words, there is no duplication within a cooperation domain. We refer to this type of caching as *exclusive caching*. Running CSC with λ set to one is same as running COOP-N-LPO. Since λ is one, there is no restriction on storing duplicated objects. In other words, $\lambda = 1$ cancels the prioritized cooperation of COOP, which causes all nodes in a partition to store the same set of most popular objects. It should now be evident that by varying λ it is possible to create different levels of object prioritization and subsequent duplications, which in turn helps reconciling between the NOA and POA performance.

7.9 Computing best POA and NOA using CSC

Maximum node level availability (NOA) is achieved when all nodes in a domain store the most popular objects and in CSC, this situation corresponds to $\lambda = 1$. In this case maximum possible NOA is limited by the cache size in each node and is equal to *f(C)*. Maximum POA is achieved when all available capacity in a domain is used to store the maximum number of objects. Thus, to maximize POA, the split factor λ must be set to 0 (which corresponds to zero duplication). The maximum POA is determined by the total number of objects we can store in a domain which is limited by cache size and number of nodes in the domain, and is equal to *f(VC)*.

7.10 Evaluation in stationary networks

In order to evaluate and compare the proposed cooperative caching mechanism, we have implemented all flavors of COOP (see Section 7.7.3) and CSC using ns2 network simulator. Simulation has been run for 100 nodes with the cache size at each node set to 100. Total number of objects is 10000, and the object popularity follows a Zipf distribution with the parameter α set to 0.8. Nodes are uniformly distributed in a 1500mx1500m area. All nodes use 802.11 MAC layer with a communication range of 250m.

7.10.1 Availability comparison between COOP and CSC

It was established in Section 7.7 that protocol COOP cannot achieve high NOA and POA at the same time. The proposed protocol CSC attempts to remove this limitation by adding the cache splitting factor λ, which together with the Domain of Cooperation (DOC) can provide a finer granularity of object duplication control. By using this finer duplication control the protocol CSC achieves high NOA and POA at the same time.

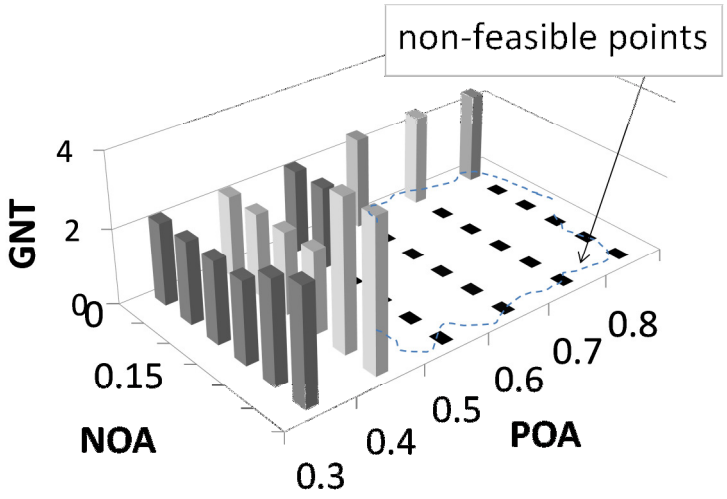


Figure 7-2: Feasible NOA-POA sets for COOP-P-LPO

Figure 7-2 shows the feasible NOA-POA points and the corresponding Traffic (GNT) values obtained from the ns2 simulation of COOP-P-LPO protocol. COOP-P-LPO is shown because it outperforms all other versions of COOP including the LRU based baseline reported in [28]. Each point in Figure 7-2 demonstrates the feasibility of a specific combination of NOA and POA as a requirement, and the corresponding GNT when the combination is feasible. For COOP, different (NOA, POA) points are generated by varying the DOC. Observe that while low (NOA, POA) points are generally mostly feasible, it is not so when either the NOA or the POA is increased. This clearly validates the logic presented in Section 7.7 that the protocol COOP cannot achieve high NOA and POA at the same time with relying only on DOC as a means for duplication control.

Figure 7-3 demonstrates CSC's NOA-POA feasibility results. Different (NOA, POA) points in this case are generated by varying the combination of DOC and the cache split factor λ.Observe how the protocol CSC is able to achieve high NOA and POA simultaneously by leveraging the additional duplication control parameter, namely, the cache split factor λ. For the experimented range of NOA and POA, only one (NOA, POA) situation, namely, (0.3, 0.8) is not achievable by CSC, whereas a large (NOA, POA) area is not feasible with COOP (see Figure 7-2). Note that the minimum possible GNT in CSC is slightly smaller than the minimum possible GNT in COOP-P-LPO uniformly across all feasible (NOA, POA) points.

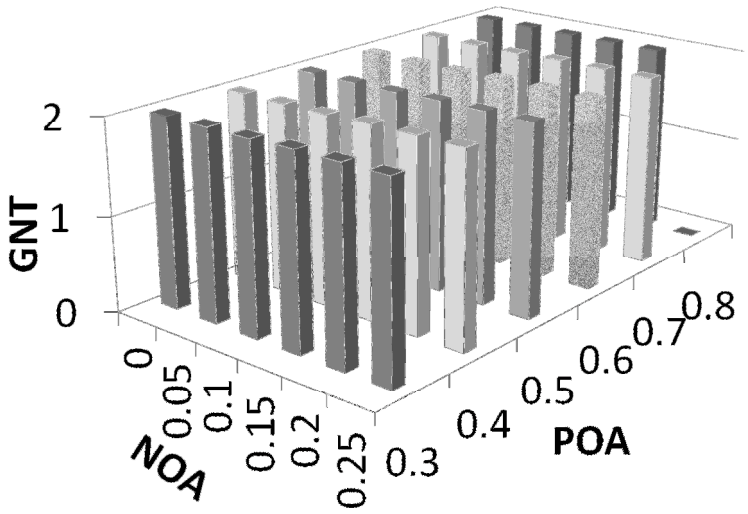


Figure 7-3: Feasible NOA-POA sets for protocol CSC

Figure 7-4 compares the feasibility region of CSC with all flavors of COOP with different cache replacement policies, namely, LRU, LPO, and RNDM. For a given replacement policy, the presented results are either for –P or –N version of COOP (whichever achieves better overall feasibility). The figure shows that the NOA-POA feasibility of CSC remains superior to all flavors of COOP irrespective of the chosen replacement policy.

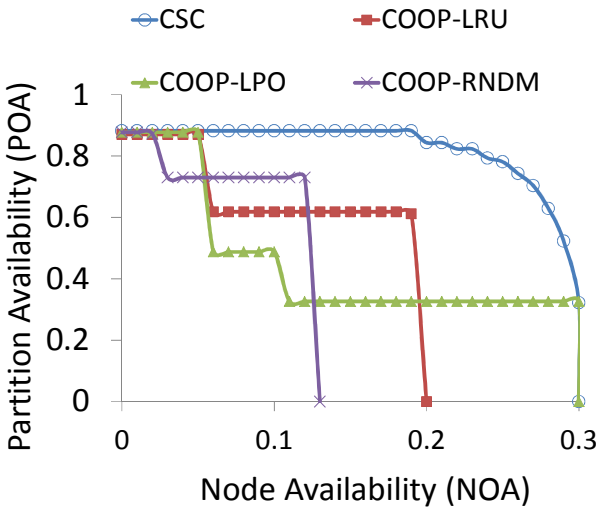


Figure 7-4: Comparative NOA-POA feasibility sets

The above results indicate that when both high NOA and NOA are needed, the proposed protocol CSC significantly outperforms all flavors of the protocol COOP, while not increasing the corresponding Traffic (GNT). In fact, as shown in Figure 7-2 and Figure 7-3, the GNT of CSC is slightly lower than that for COOP at the feasible (NOA, POA) points. This property of CSC was found to be valid for different cache size and also for different Zipf parameters.

7.10.2 Individual characterization of COOP

Node Object Availability (NOA)**:** As defined in Section 7.6 , for a given Domain of Cooperation (DOC), using object prioritization reduces duplication by storing more unique objects within the DOC region. As shown in Figure 7-5, since lower object duplication leads to less likely local hits, the NOA is generally lower with prioritization turned on for all flavors of COOP. With prioritization on, since increasing DOC reduces the level of object duplication, the quantity NOA reduces with increasing DOC. With prioritization off however, DOC does not have major impact on NOA because in this case the level of object duplication is already *high and* is decided primarily by the local replacement policies such as LRU, LPO or RNDM.

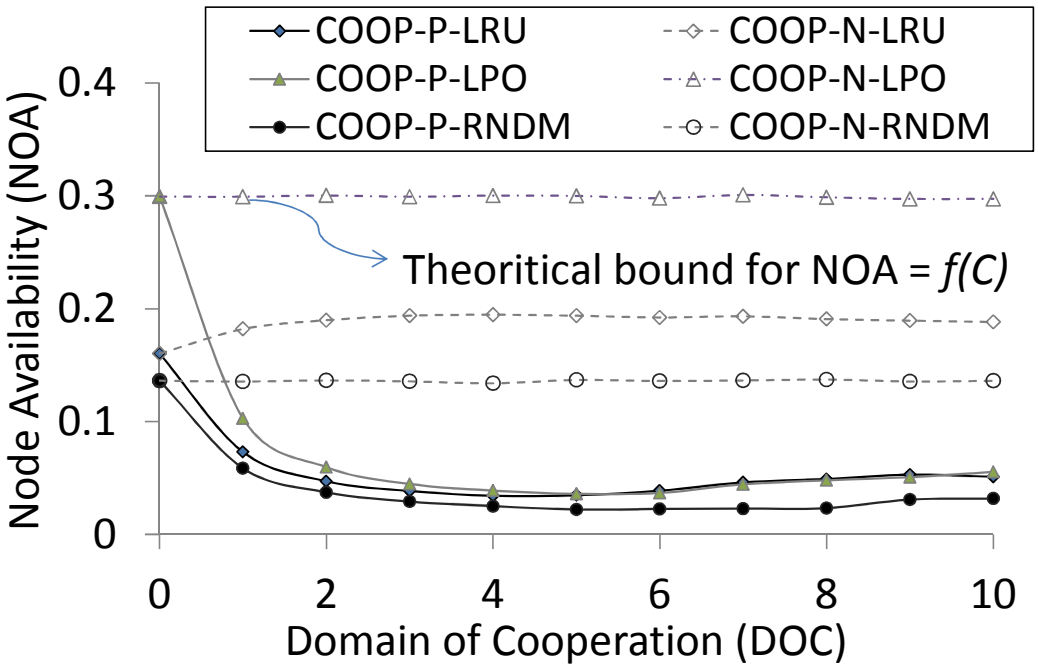


Figure 7-5: Impacts of DOC on NOA

Partition Object availability (POA): As shown in Figure 7-6, unlike in NOA, the POA demonstrates a general increasing trend with increasing DOC. With higher DOC, a node searches a bigger network area for an object, thus increasing the likelihood of finding it in the partition. With prioritization turned on, the impacts of DOC are more pronounced because prioritization itself helps reducing duplication. Therefore, to achieve high availability object prioritization must be turned on which in turn deteriorates NOA.

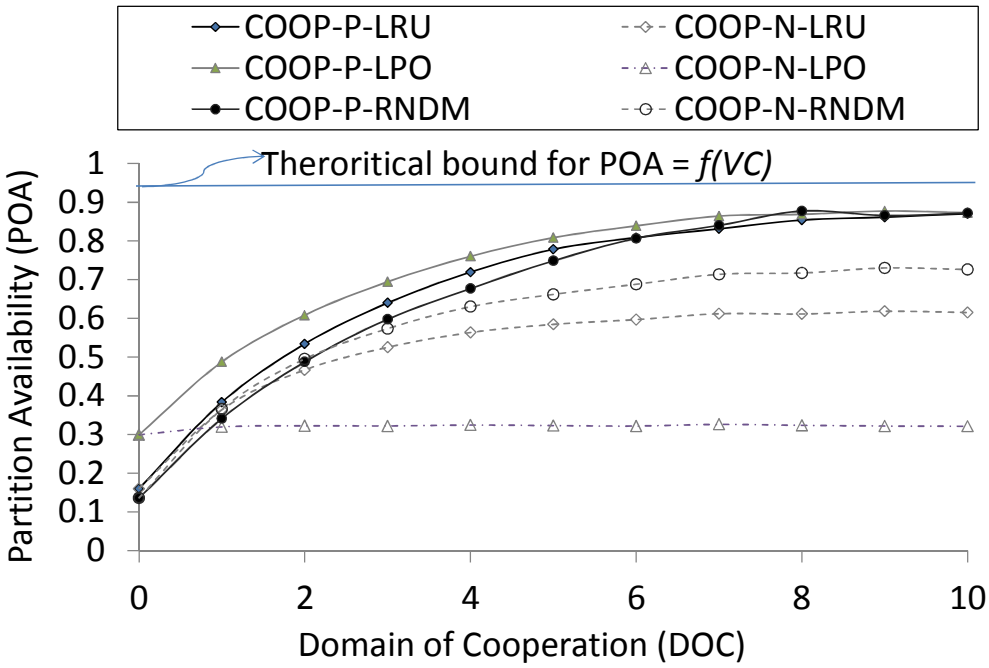


Figure 7-6: Impacts of DOC on partition availability

7.10.3 Individual characterization of CSC

As explained in Section 7.8, DOC and cache split factor λ are the two parameters which control the object duplication level, thus determining the availability performance for the proposed CSC mechanism. Figure 7-7 demonstrates the impacts of DOC and λ on the Node Object Availability (NOA). Observe that NOA depends a great deal on λ. larger λ values cause higher levels of object duplications, leading to larger node level availability due to increasing local hit rates. With smaller λ, more unique objects are stored within the cooperation domain, leading to lower node level local hit rates, thus causing lower NOAs. In CSC, the set of duplicated objects in all nodes are the same and this set does not change as we increase DOC. In other words, varying DOC does not change NOA as it is shown in Figure 7-7.

The impacts of DOC and cache split factor λ on the Partition Object Availability

(POA) are reported in Figure 7-8. As expected, with lower λ, since more unique objects are cached within a cooperating domain, the likelihood of finding an arbitrary object within the partition increases. This explains the higher POA numbers for lower λ values. As for the impacts of DOC, with higher DOCs, a node searches a bigger network area for an object, thus increasing the likelihood of finding it in the partition – leading to higher POAs.

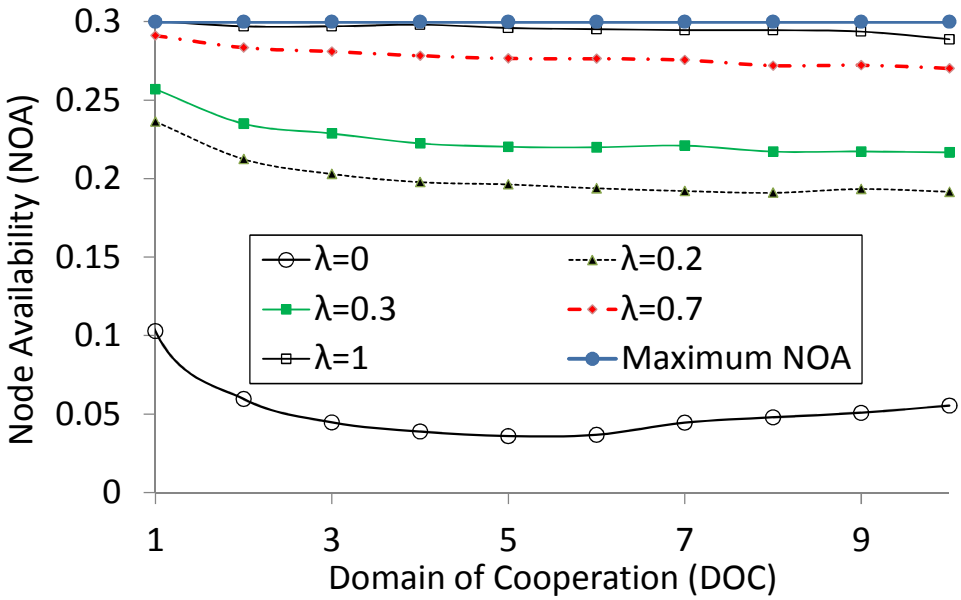


Figure 7-7: Impacts of DOC and λ on NOA in CSC

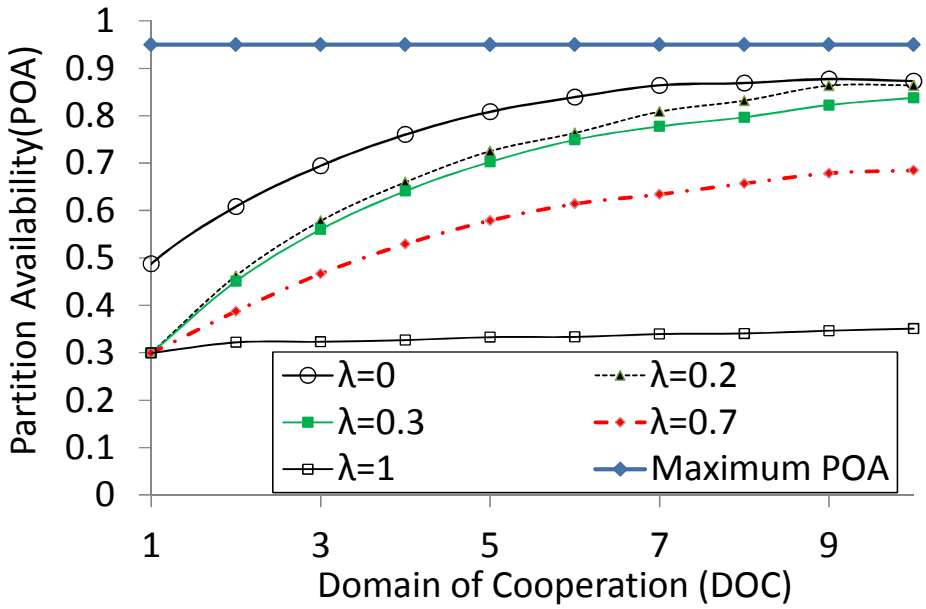


Figure 7-8: Impacts of DOC and λ on POA in CSC

From the NOA and POA results in Figure 7-7 and Figure 7-8, it can be observed that unlike in COOP, the proposed CSC caching protocol can achieve high NOA and POA simultaneously by carefully adjusting the two object duplication control parameters DOC and λ. The impacts of this advantage of CSC over COOP have already been shown in terms of the NOA-POA feasibility landscapes in Figure 7-4.

7.11 NOA-POA Feasibility in Mobile networks

In this section we relax the stationary partition assumption which was used for all the results presented in Section 7.10. The same baseline parameters are used in ns2, with human mobility modeled as random waypoint with an average node speed of 1.2m/s (which is close to human speed) and pause time 300 seconds. Impacts of different pause times, representing different levels of mobility, are also evaluated.

Figure 7-9 shows the feasible NOA-POA points and the corresponding Traffic

(GNT) values for the COOP-P-LPO protocol, which provides the best feasibility results among all flavors of COOP. Observe that like in the stationary case, while low (NOA, POA) points are mostly feasible, it is not so when either the NOA or the POA is increased. The main difference between this graph and that of stationary network (i.e. Figure 7-2) is that because of mobility, DOC and its resulting object prioritization have less impact on NOA and POA. When nodes are mobile, it's difficult to prioritize unique objects over duplicated objects since there is always a minimum degree of undesired duplication in the network. This explains why achieving high values of POA is not possible for COOP based schemes. Note that because of using object prioritization achieving high values of NOA also is not feasible.

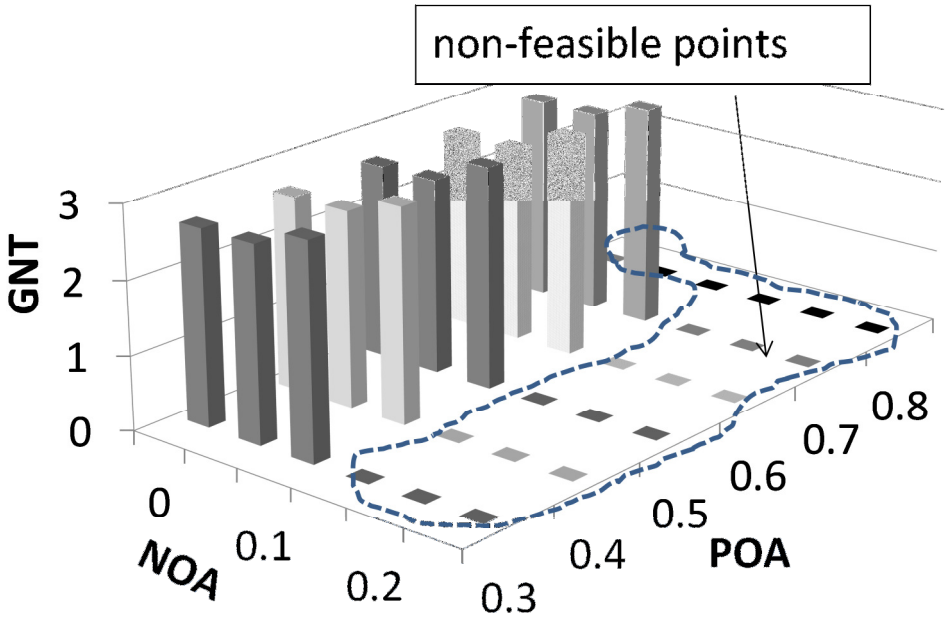


Figure 7-9: Feasible NOA-POA sets for COOP-P-LPO

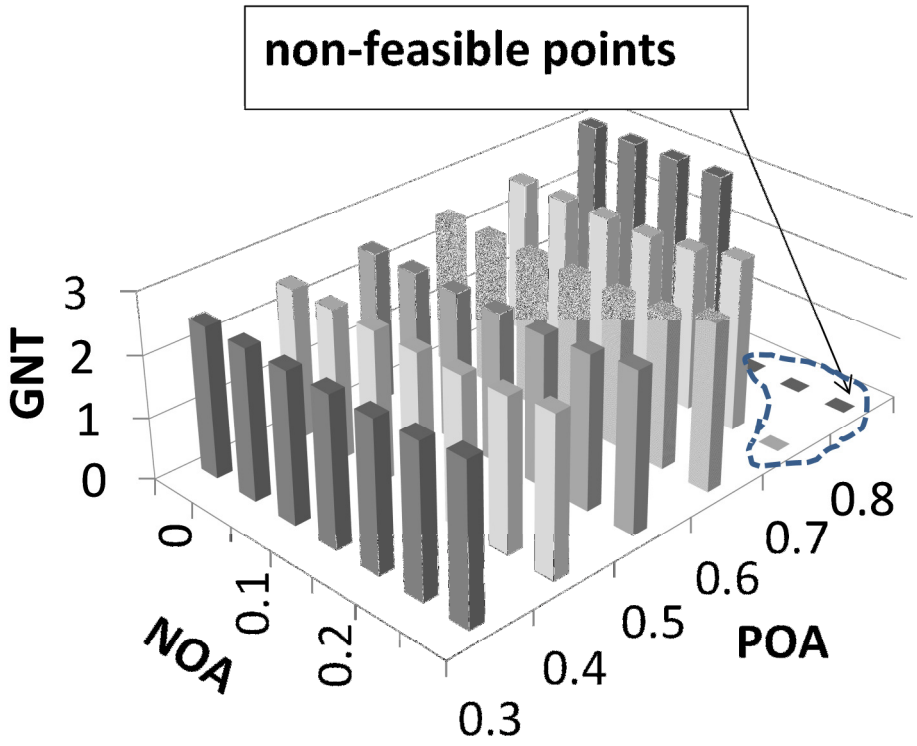


Figure 7-10: Feasible NOA-POA sets for CSC

The NOA-POA feasibility points for the proposed CSC scheme are reported in Figure 7-10. Comparing Figure 7-3 and Figure 7-10, it can be observed that the NOA-POA feasibility performance of CSC does not change appreciably when mobility is introduced. Different (NOA, POA) points in this case are generated by varying the combination of DOC and the cache split factor λ. Like in the stationary case, the minimum possible GNT in CSC is slightly smaller than the minimum possible GNT in COOP-P-LPO across all feasible (NOA, POA) points.

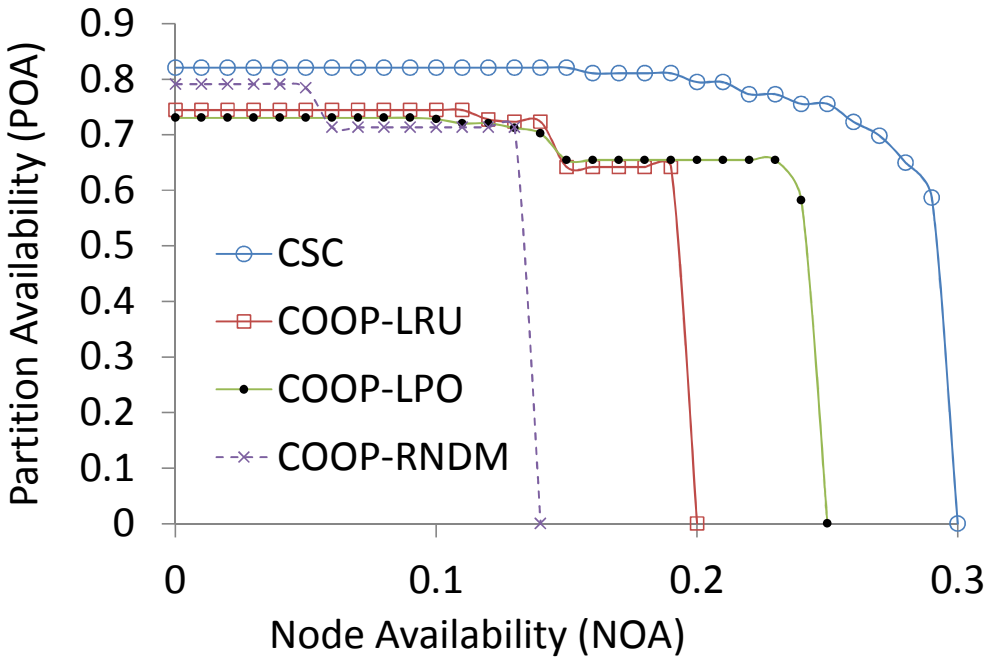


Figure 7-11: NOA-POA feasibility in a mobile network

Figure 7-11 compares the feasibility region of CSC with all flavors of COOP with different cache replacement policies. The figure shows that the NOA-POA feasibility of CSC remains superior to all flavors of COOP irrespective of the chosen replacement policy. The above results indicate that in addition to stationary networks, the proposed protocol CSC significantly outperforms all flavors of the protocol COOP under mobile networks. Moreover, as shown in Figure 7-9 and Figure 7-10, the GNT of CSC is slightly lower than that for COOP.

The impact of the degree of mobility is evaluated by changing the pause time between two subsequent moves in ns2 random waypoint mobility. Figure 7-12 shows that for mobile network the maximum POA is slightly lower than that of a stationary network. The main reason is under mobile network there is always a minimum degree of undesired duplication in a domain which reduces the maximum possible POA. For example when a node gets isolated, it has to download all objects from the CP's server when they are not

available in the local cache. Among those object node will store the most popular objects. Later when a node gets connected to a bigger partition, it has a set of duplicated objects and the number of these objects is more that λC. This extra undesired duplication reduces the total number of unique objects stored in the partition and, in turn, reduces the POA. As discussed in Section 7.8.3, in CSC the requester notifies all nodes who maintain a duplicate object so that those nodes can remove the duplicated object from their cache to accommodate a new unavailable object in the partition. This mechanism alleviates the impact of undesired duplication; however, it cannot completely fix it.

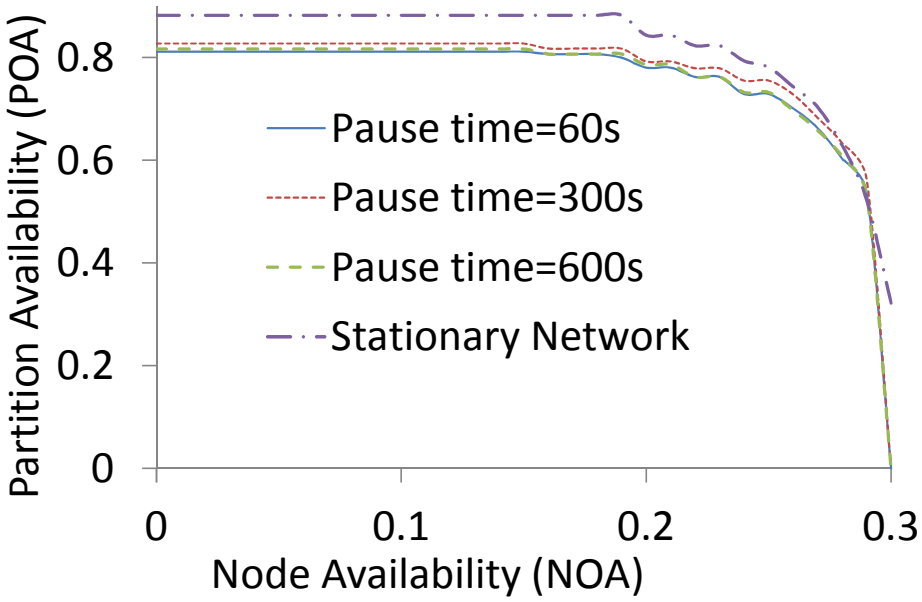


Figure 7-12: Impacts of pause time on NOA-POA feasibility

7.12 **Evaluation of Generated Network Traffic**

Figure 7-13 shows the effects of DOC on GNT for the COOP family of protocols. GNT is a function of NOA, POA, and the topological descriptors C_d and β. The quantity C_d is the average hop-count between a node and Internet gateway and the quantity $\beta.C_d$ is the average hop-count between any two nodes in a domain.

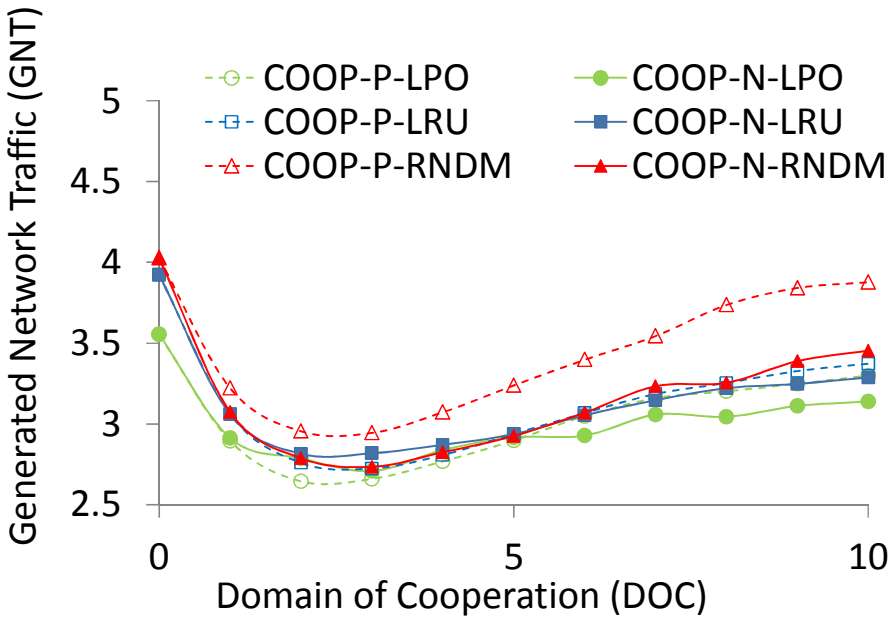


Figure 7-13: Impacts of DOC on GNT in protocol COOP

In Figure 7-13, it can be observed that the GNT first decreases with increasing DOC and then after an optimal point, it starts increasing back up. The reason is initially, by increasing DOC, a percentage of the requested objects are found within the domain in a distance smaller than C_d. In a small domain, the average hop-count between the requester and found object in the domain is significantly smaller than the average hop-count between a node in the domain and the gateway nodes (i.e. $\beta << 1$).

For larger domains, however, the distance between the requester and the found object increases and get closer to the average distance between a node and the gateway node (i.e. $\beta \sim 1$). This causes the GNT to go up again.

Figure 7-14 reports the generated traffic as a function of the two duplication control parameters DOC and λ in CSC. Similar to COOP, there is an optimal DOC at which GNT is minimized. Furthermore, there is second point of optimality for λ at optimal

DOC which provides the minimum GNT among all CSC policies using the optimal DOC. This explains as to why GNT for CSC is slightly smaller than that of COOP at all feasible points (as shown in Figure 7-9 and Figure 7-10).

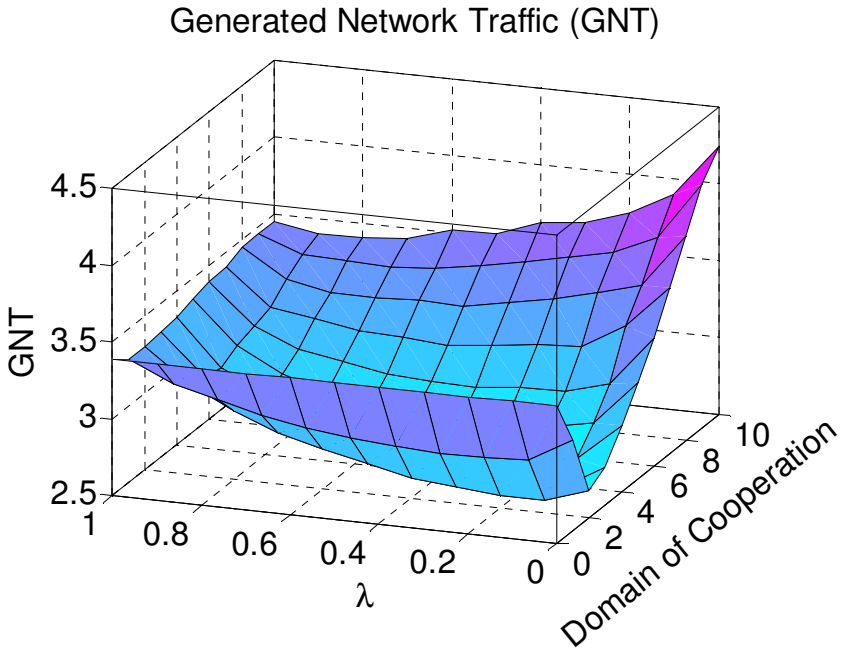


Figure 7-14: Impacts of DOC and λ on GNT in CSC

7.13 **Summary and Conclusion**

We have developed a cooperative object caching mechanism for providing high network and node level object availability in Mobile Social Wireless Networks. This is achieved by using a novel cache partitioning method which can be used for fine grain object duplication control within isolated network partitions. It was demonstrated that by using this cache partitioning strategy, the proposed mechanism is able to outperform the existing schemes while reducing network traffic. The above conclusion holds for both stationary and mobile networks.

Chapter 8 : COOPERATIVE FIREWALLING IN MANETS

8.1 Introduction

Wireless networks, such as Mobile Ad Hoc Networks (MANETs) and wireless mesh networks, have become an integral part of the Internet infrastructure. Unwanted traffic, which constitutes 2 to 12 percent of the Internet traffic, wastes significant network bandwidth and the power of resource-constrained wireless nodes [64]. For example, worm attacks are one the most severe cyber threats for MANETs in which worms with arbitrary payload can saturate the networks in a matter of seconds [155].

On the Internet, firewalls are widely deployed on the border of private wired networks to stop unwanted traffic to and from the public network. However, unlike wired networks, it is difficult to deploy firewalls for wireless mobile networks because each wireless mobile node often manages itself and therefore a central firewall is often unrealistic to reach and impractical to enforce. Furthermore, due to the mobility and topology dynamism, wireless mobile networks often lack the concept of private networks and therefore have no clear line of defense. To defend against malicious attacks, each wireless node has to implement the firewall functionality by itself. However, discarding unwanted traffic at destination nodes in wireless networks leads to significant waste of scarce resources, such as bandwidth and power, used by intermediate nodes to forward unwanted traffic.

In this chapter we use the proposed cooperative caching scheme as a solution for discarding unwanted packet in the MANETs. A firewall rule is considered as an object and unwanted traffic is considered as object requests. Knowing the restricted capacity of

mobile node, only a limited number of rules can be stored in each node, the main question then is how to place firewall rules in the network to discard the maximum number of unwanted packets in the network.

8.2 Technical Challenges

To discard unwanted traffic before reaching destinations, for each wireless node, we need to distribute its firewall rules to other nodes. However, distributing firewall rules in wireless networks is a technically challenging problem. First, the topology in wireless mobile networks is dynamic and so are the forwarding paths. Thus, for the firewall rules that a node wants other node to enforce, it is difficult to identify which nodes these rules should be sent to. Second, the number of rules that a wireless node can handle is rather limited due to resource limitations. Thus, for the firewall rules that a node receives, it is difficult to decide which rules should be admitted and enforced given its resource constraints.

8.3 Using Cooperative Firewall Rule Caching

In this section we propose a distributed firewalling scheme for wireless mobile networks where nodes collaboratively discard unwanted packets for each other. We address the first challenge of topology and path dynamism by embedding firewall rules within routing messages and distributing a node's rules along the paths that the node receives unwanted traffic so that unwanted packets can be discarded before they reach the node. Coupling rule distribution with routing update messages allows us to find the paths that unwanted traffic are received and therefore distribute rules along them. For proactive routing protocols, where each node periodically sends routing messages to other

neighbors, firewall rules are sent out along proactive routing messages so that the nodes receiving the routing messages can enforce these rules. For reactive routing protocols, when a node wants to send a packet, the header of the packet is included in the route request message. If the packet is unwanted, the destination node includes the rule for discarding this packet in the corresponding route reply messages and hereby notifies all intermediate nodes in the path from the packet source to the packet destination. We address the second challenge of resource limitations by each node enforcing portions of its received firewall rules based on rule admission policies and replacing obsolete rules by new rules based on rule replacement policies. We use *Split Cache* replacement policy as described in Chapter 3 and also we propose another heuristic based rule admission and replacement policies to maximize the number of unwanted packets discarded before reaching their destinations.

8.4 Rule Distribution Framework

In this section, we present our rule distribution scheme addressing both rule exporting (*i.e.*, determining which rules to export to neighbors and how to export them) and rule importing (*i.e.*, determining which received rules to be enforced).

8.4.1 System Model

The wireless network is formed by mobile nodes without any prior contact, trust or authority relation. The nodes are able to communicate with each other using different multi-hop routing protocols. Since the nodes can be mobile, the topology of the network may change frequently over time. Mobile nodes are usually resource constrained in energy, bandwidth, storage, memory and computational ability. Each mobile node may

have some firewall policy that specifies what packets it does and does not want to receive from other nodes. The firewall policy is represented by an Access Control List (ACL) consisting of a sequence of rules. Each rule has a predicate over some packet header fields and a decision (*i.e.*, action) to be taken for the packets that match the predicate. The decision of a rule is typically `accept` (*i.e.*, permit) or `discard` (*i.e.*, deny). ACL rules are often overlapping and decisions are made based on the first match semantics (*i.e.*, the decision that an ACL makes for a packet is the decision of the first rule that the packet matches in the ACL). Thus, for rule distribution purposes, we need convert each overlapping ACL to an equivalent non-overlapping ACL. Each node only exports discard rules (*i.e.*, the rules whose decision is `discard`) and they are stored in a table called Export-Policy Table (EPT). To ensure that the rules exported by a given node can only be matched by packets that destined to this node, the destination field of each rule in the node's EPT must be the address of the node. Accordingly, for each node, we store the rules that it receives and it wants to enforce in a table called Import-Policy Table (IPT). Before forwarding a packet, a node checks the packet header against IPT rules and discard the packet if it matches a rule in the IPT. Figure 8-1 shows a simple example network where node D distributes the rules in its EPT, as shown in Table 8-1: Export-Policy Table of node D, to other nodes. In this example network, for three rules of D, we suppose node A admits rule r_1 and r_2, B admits rule r_1, C admits rule r_2, and S admits nothing. Suppose node S sends a packet with destination port 80 to D, it is forwarded by B but discarded by A before reaching D because this packet matches rule r_2 in A's IPT.

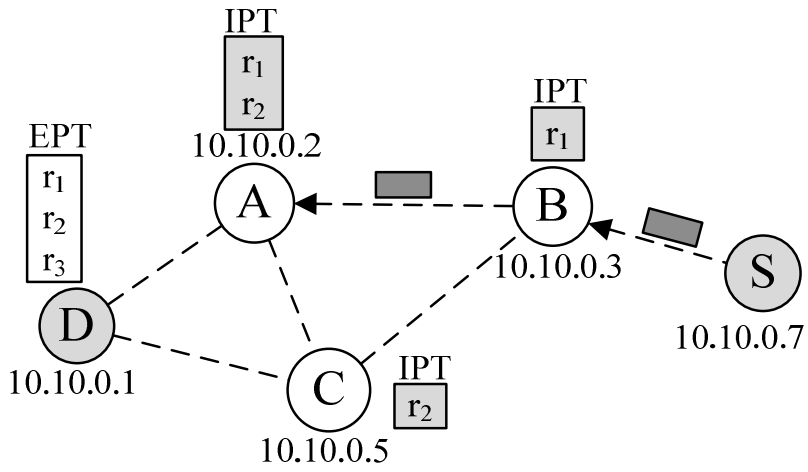


Figure 8-1: Rule distribution on an example network

Rule	Src IP	Dest IP	Src Port	Dest Port	Protocol	Action
r_1	10.10.0.5	10.10.0.1	*	8080	TCP	discard
r_2	10.10.0.7	10.10.0.1	*	80	*	discard
r_3	10.10.0.3	10.10.0.1	*	*	*	discard

Table 8-1: Export-Policy Table of node D

8.5 Rule Exporting

8.5.1 Constructing Export-Policy Table

To construct EPTs, we need to convert overlapping ACLs to equivalent non-overlapping ACLs. We perform this conversion using Firewall Decision Diagrams (FDD), a tree-based data structure for representing ACLs [156]. An FDD is a directed acyclic graph (DAG) with the following properties: (1) It has exactly one root node. (2) Each non-terminal node represents a packet field and each terminal node represents a decision. (3) A directed path from the root to a terminal node is called a *decision path*. The node labels on every decision path are unique. (4) Each edge is labeled with a non-empty set of integers within the domain of the field that labels the node where the edge is originated. (5) The sets of integers that label the outgoing edges of a node are non-overlapping. The union of these sets equals to the domain of the field that labels the node.

After converting an overlapping ACL to an equivalent FDD, we can generate an equivalent non-overlapping ACL from the FDD by generating one rule per decision path. Deleting accept rules from the non-overlapping ACL yields the EPT. Figure 8-2 illustrates the process of calculating an EPT from an overlapping ACL. Figure 8-2(a) shows a toy example of a two dimensional ACL whose rules are overlapping. We first convert this ACL to an equivalent FDD shown in Figure 8-2(b), and then using this FDD, we extract non-overlapping discard rules to construct the EPT in Figure 8-2(c).

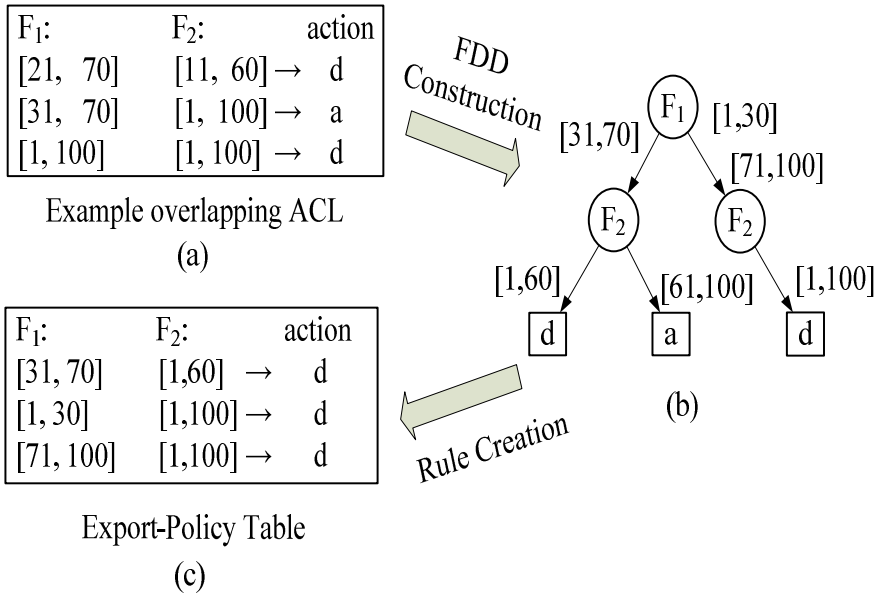


Figure 8-2: EPT Construction using FDDs

8.5.2 Rule Distribution

We distribute rules in EPTs along with routing messages. Below, we discuss our rule distribution scheme based on two types of routing protocols used in wireless mobile networks: proactive protocols and reactive protocols.

8.5.2.1 **Rule Distribution with Proactive Routing Protocols**

In proactive routing protocols such as OLSR [157] and DSDV [158], each node periodically sends routing updates to its neighbors to keep the nodes' routing table consistent. Such protocols are suitable for stationary networks such as wireless mesh networks and the wireless networks that are communication intensive. Using such protocols, for each discard rule in the EPT of a given node, this node keeps track of the hit rate of the rule, *i.e.*, the number of received packets that match the rule per unit time; once the hit rate of a rule exceeds a threshold, the node sends out the rule piggybacked on its routing messages. A rule r is associated with two variables, *hit rate* denoted h and *admission distance* denoted d. When a node receives a rule and admits it in its IPT, it resets the admission distance value of this rule to be its distance to the node that this rule is originated from and then forwards the rule to the next hop. To control the number of hops a rule traverses, each rule has a Time to Live (TTL) value. A rule is not forwarded if its TTL value is equal to 0. Figure 8-3 shows an example scenario that illustrates the above rule distribution process.

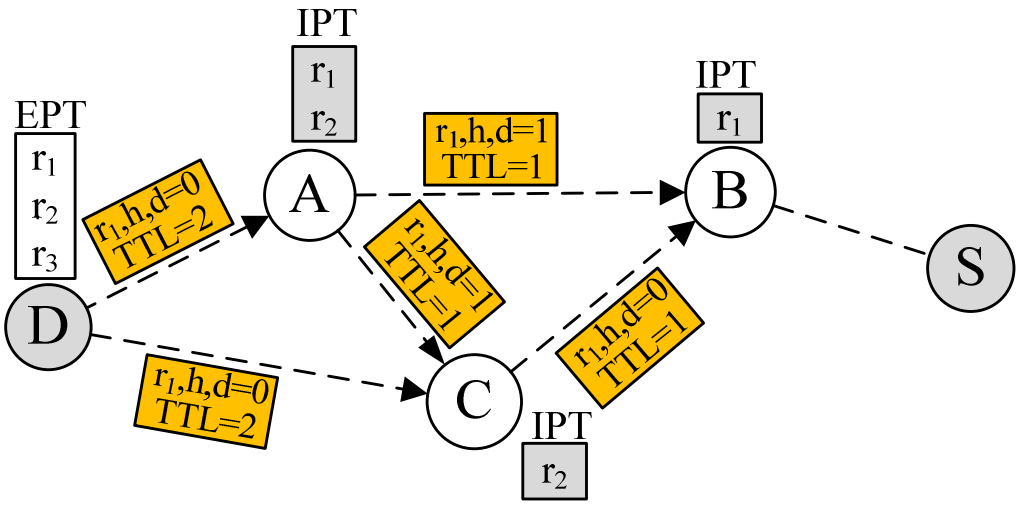


Figure 8-3: Rule distribution with proactive routing protocols

8.5.2.2 **Rule Distribution with Reactive Routing Protocols**

Reactive routing protocols, such as AODV [138] and DSR [159], are suitable for wireless networks that are mobile and that are not communication intensive. In such protocols each node broadcasts a Route Request (RR) message to find a path to the destination node that it wants to send messages; upon receiving an RR message, a Route Reply (RRep) message is sent either by an intermediate node (if the node knows a path from itself) to the destination or the destination node. The RR messages include source and destination address to discover the route to the destination. In our rule distribution scheme, we modify RR messages so that they also include other packet header fields such as destination port, source port, and protocol type. Before the destination node or an intermediate node replies to an RR message with an RRep message, it checks the packet header fields included in RR message against the rules in its IPT. The destination node, however, checks the packet header against its EPT. If the packet header does match any of the rules, the node includes the corresponding rule r with its hit rate h and admission distance $d = 0$ in the RRep message and sends it over to the source. Note that hit rate of rule r is updated each time a packet header matches rule r. When an RRep messages travels back to the source node, it informs all intermediate nodes in the path about rule r. Upon receiving the RRep message, the source node or the intermediate nodes can decide whether to import the rule based on its hit rate and admission distance. Note that if a node decides to import rule r to its IPT, it must reset d in RRep message to be its distance to the destination before forwarding the rule to the next hop. Figure 8-4 shows an example scenario that illustrates the above rule distribution process. In this example, node S wants to send an HTTP request to node D over destination port 80. To find the path to node D,

node S sends an RR message including source port (SP=1111), destination port (DP=80), and protocol (P=TCP). As rule r_2 in the EPT matches the packet header fields in the RR message, node D includes rule r_2 along with its h and $d = 0$ to the RRep message and sends it over to node S. As node A adds this rule in its IPT, it updates d to 1. Finally, node S realizes from the RRep message that its packet will be discarded by node D. Hence, node S stops sending any HTTP request.

As reactive routing protocols usually use route caching, a node may receive unwanted packets without receiving the corresponding RR message. Thus, only relying on reactive approach to distribute rules may not be sufficient. Therefore, we can use the proactive approach along with the reactive approach.

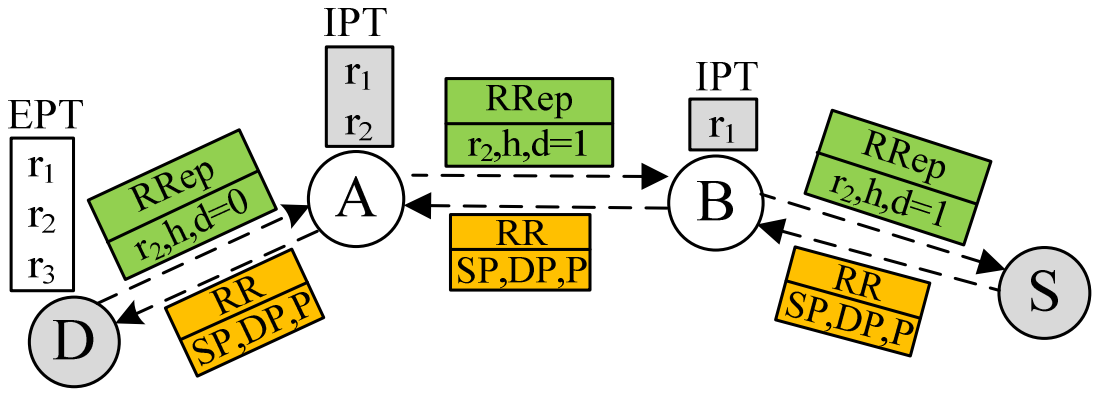


Figure 8-4: Rule distribution with reactive routing protocols

8.6 **Rule Importing**

When a node receives a rule, it decides whether to admit the rule to its IPT based on its admission and rule replacement policies. Rule admission policy is based on multiple factors such as the rule's originator, hit rate, and admission distance. For instance, a node may ignore a received rule if its hit rate is less than a prescribed threshold. If a node decides to store a new rule while its IPT is full, it must evict another

rule from its IPT to accommodate the new rule based on certain replacement policy. In Section 8.11, we introduce two replacement policies to optimize rule distribution based on our system performance metrics.

8.7 Policy Table Consistency

When the ACL rules of a node are modified, probably by its administrator, the EPT of the node has to be recomputed. The new EPT rules may be different from the old EPT rules, which may have been exported and admitted by other nodes. To avoid such inconsistency, we propose two mechanisms that all nodes can employ to remove the admitted rules originated from a particular node: *rule revocation* and *rule expiration*. In the rule revocation mechanism, a node can broadcast a revocation message that contains all the rules that this node wants to revoke. In the rule expiration mechanism, each rule in an IPT has an expiration time, which is set based on the rule lifetime. The rule is deleted when it expires. These two mechanisms can be used together.

8.8 Security Analysis

8.8.1 Threat Model

Distributing rules in an insecure network in which mobile nodes have low physical security and can be easily stolen or compromised by an adversary raises some serious security issues. First, an adversary node can impersonate a victim node and send forged rules to deceive intermediate nodes to discard the legitimate traffic destined to the victim node. Second, an adversary node can modify forwarding rules such that they cause partial or complete unreachability of the victim node from the rest of the network. Third, an adversary node can modify the rule lifetime to extend or shorten the rule effectiveness,

which leads to unexpected reachability problems. Fourth, an adversary node can store and resend a rule to cause unexpected reachability problems (aka replay attack). Fifth, an adversary node can generate or store and resend a revocation message to all the nodes in the network, which disrupts rule inconsistencies and causes unexpected reachability problems.

8.8.2 Securing Rule Distribution

To address the first three security threats, we need to employ a set of security mechanisms that provides *rule sender authentication* and *rule integrity*. Such security mechanisms require pre-distributed symmetric keys or a reliable Public-Key Infrastructure (PKI). While mesh networks that have solid network topology often have pre-established security infrastructure, MANETs may not have such infrastructure due to node anonymity or network unreliability caused by node mobility, sparse connectivity, etc. Thus for mesh networks, we assume that the network has pre-distributed symmetric keys or PKI, in which each node knows which key (or public key) is associated to which IP address. And for MANETs, we assume that the nodes are auto configured using secure auto configuration scheme proposed by Wang *et al.*in [160]. In this scheme, each node first computes its public key and accordingly chooses an IP address, which is the hash of its public key. Using such a scheme or similar ones that provide key-to-IP mapping is useful for rule sender authentication. Hence, using a PKI system with a key-to-IP mapping, a rule sender node creates and distributes a message containing the rule and its attributes such as hit rate, admission distance, TTL, lifetime and a nonce as well as the rule digital signature and the node's public key. An example exported rule message is shown in Figure 8-5.

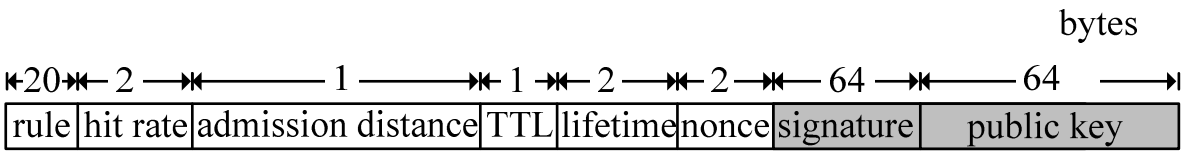


Figure 8-5: The format of a secure rule message

On the other hand, when a node receives a rule, it first checks whether the public key belongs to the rule sender. It then verifies the message digital signature for message integrity and authentication. Finally, it ensures that the rule's destination field is the rule sender IP address. This verification is to confirm that the rule sender is not impersonated and the rule as well as its attributes is not modified. Once the rule is verified, it is added to the node's IPT with its corresponding nonce.

To address the replay attack (*i.e.*the fourth security threat), each rule has a nonce. For reactive routing protocols the nonce is the RR message sequence number for which the rule is exported. Thus, as each rule is assigned to an RR message, an adversary cannot replay a rule with no corresponding RR message. For the proactive routing protocols, on the other hand, the nonce is a timestamp, which is the time that the rule is exported. Let t_s be the timestamp and t_r be the receiver node clock. The receiver node accepts a rule if $|t_r - t_s| \leq \epsilon$. Using this condition, we bound the replay attack time window ($|t_r - t_s|$) to a constant value ϵ which is set by the receiver node. If the nodes are even loosely time synchronized, ϵ can be chosen to be very small; otherwise, it needs to be chosen large enough to create a balance between unexpected unreachability and unwanted packet filtering.

To address the fifth security threat and revoke a rule securely, we use the unique nonce corresponded to each rule to avoid replaying revocation messages. Thus, to send a revocation message, the destination node first recreates the rule message with its original

nonce but with lifetime set to zero. It then signs and broadcasts the message in the network. As the rest of the nodes receive the revocation message, they delete the rule with is corresponding nonce from their IPT. By this technique, an adversary cannot generate a revocation message because it must be signed by the rule sender (which is the destination field of the rule). Also, an adversary cannot store and resend a revocation message because each revocation message is for only one rule with its corresponding nonce. Note that the hit-rate and admission distance fields for the revocation messages are set to zero.

8.9 Performance Model

8.9.1 Protocol Overhead

Due to network resource constraints, the main objective of protocol design is to minimize the computational and transmission overhead of IPT computation and rule distribution. As the number of firewall rules for a given node is usually very small, finding non-overlapping rules for EPT and distributing them occasionally requires insignificant amount of resources. More precisely, using reactive routing messages, the rules are piggybacked by RR and RRep messages, which only increases their sizes up to 5 and 156 bytes respectively, and no extra messages are required. Similarly, using proactive routing messages, a node can either piggyback new rules on its own routing messages, which again results in a very small overhead, or it can send an explicit broadcast message for announcing new rules. Recall that using reactive routing protocol, a demand for an unwanted packet triggers rule distribution, and for proactive routing protocol, the number of unwanted packet per unit time must exceed a prescribed threshold before any new rule is dispatched. Thus, rules are mostly distributed when the

network is seemingly under attack. Hence, compared to the number of unwanted packets that can be stopped in the path, such small overhead of rule distribution is negligible.

8.9.2 Performance Metrics

We define two performance metrics:(1) Packet Discard Ratio (PDR) that represents the percentage of unwanted packets discarded in the path before reaching the destination, and (2) Forwarding Cost Ratio (FCR) that represents the expected value of the portion of its path that an unwanted packet is forwarded before being discarded. The ultimate goal is to maximize PDR for destination node and to minimize FCR for the entire network. However, achieving both performance goals at the same time is difficult because the size of IPTs is limited. To maximize PDR, we need to maximize the number of unwanted packets that are discarded before reaching their respective destinations. Thus, we want to distribute the maximum number of unique rules on the nodes along a path that match unwanted packets. However, as the size of a node's IPT may be smaller than the number of the rules that the node receives, the rules needs to be distributed on the nodes with the minimum amount of rule duplication. On the other hand, to minimize FCR, we need to maximize the number of the unwanted packets that are discarded at sources (*i.e.*number of forwarding hops is zero). This requires each node to store high hit rate rules in its IPT that in turn results in more rule duplications. To address both performance metrics at the same time, we define overall cost $\mathcal{C}$ as a weighted function of PDR and FCR as follows:

$$\mathcal{C} = \gamma \times (1 - PDR) + (1 - \gamma) \times FCR \quad (8\text{-}1)$$

In Eqn. 6-1, $\gamma \in [0,1]$ is the weight coefficient factor that balances between PDR and FCR. For example, when $\gamma = 1$, the overall cost only depends on PDR, whereas when $\gamma = 0$, the overall cost only depends on FCR.

8.9.3 Analytical Model

Given a network with n nodes, we first model collaborative firewalling for a single destination node that receives unwanted packets from $n-1$ nodes. We then extend this model to all nodes in the network.

Suppose node i sends λ_i unwanted packets per unit time to the destination node D. We assume the destination node D has N rules $r_1, \ldots, r_N$ in its EPT to export. The *popularity* of a given rule is defined as the percentage of unwanted packets that match against the rule. The popularity of i-th rule is denoted by p_i and is calculated as $p_i = \frac{h_i}{n\sum_{i=1}^{n}\lambda_i}$; where h_i is the number of unwanted packets that match against i-th rule. For simplicity, we assume rule popularity follows a power law distribution. We model p_i by Zipf distribution with parameter α and Ω (*i.e.*$\Omega = \sum_{i=1}^{N} i^{\alpha}$) as follows:

$$p_i = \frac{1}{i^{\alpha}\Omega} \tag{8-2}$$

Note that $r_1, \cdots, r_N$ are sorted where r_1 and r_N have the highest and the lowest popularity, respectively. Let $F_{iD}(t) =< s_0, s_1, \cdots, s_m >$ be the sequence of nodes in the path from the i-th node in the network to destination D. To be able to enumerate the nodes in the path, we use s_0 to represent the i-th node in the network and s_m is the closest node to the destination. Let $R_j(t)$ be the set of rules stored at s_j at time t and $P(R_j(t))$ be the probability that a given unwanted packet matches against one of the rules in $R_j(t)$. As $R_j(t)$ contains non-overlapping rules, $P(R_j(t))$ is calculated as follows:

$$P(R_j(t)) = \sum_{r \in R_j(t)} p_r \tag{8-3}$$

Let $PDR_{ij}(t)$ be the PDR of unwanted packets sent from i to j at time t. Thus,

$$PDR_{iD}(t) = P(\bigcup_{\forall s_j \in F_{iD}(t)} R_j\,(t)) \tag{8-4}$$

Using Eqn. 6-4, we can compute the average PDR for all the unwanted packets that reach destination D. The average PDR of all unwanted packets from different sources denoted by $\overline{PDR}_D(t)$ is calculated as follows:

$$\overline{PDR}_D(t) = \frac{\sum_{i=1}^{n}(\lambda_i \times PDR_{iD}(t))}{\sum_{i=1}^{n}\lambda_i} \tag{8-5}$$

And the total average PDR for the destination node D in time period T is calculated as follows:

$$\overline{PDR}_D = \int_0^T \overline{PDR}_D\,(t)dt \tag{8-6}$$

Based on the definition of FCR, it is the ratio of number of times that a packet has been forwarded to the path length between source node and destination node. For example, FCR of an unwanted packet discarded at source node is zero, and it is 1 if it is discarded at the destination. Let $FCR_{ij}(t)$ be the FCR of unwanted packets sent from i to j at time t. We can calculate $FCR_{SD}(t)$ for a simple example network in Figure 8-1, where $F_{SD}(t) =< S, B, A >$, as follows:

$$FCR_{SD}(t) = 1/3\{0 \times P(R_0(t)) + 1 \times P(R_1(t) - R_0(t)) + 2$$

$$P(R_2(t) - R_1(t) - R_0(t)) + 3(1 - PDR_{SD}(t))\}$$

$R_0(t)$, $R_1(t)$, and $R_2(t)$ are the set of rules for nodes A, B, and S. The generalized equation for $FCR_iD(t)$ is as follows:

$$FCR_{iD}(t) = \frac{\sum_{\forall s_j \in F_{iD}(t)}\left(j \times P\left(R_j(t) - \bigcup_{k<j} R_k\right)\right)}{|F_{iD}(t)|} \tag{8-7}$$

Using Eqn. 6-7, we can compute $FCR_D(t)$, the FCR for all unwanted packets that reach destination D as follows:

$$FCR_D(t) = \frac{\sum_{i=1}^{n} \lambda_i \times FCR_{iD}(t) \times |F_{iD}(t)|}{\sum_{i=1}^{n} \lambda_i \times |F_{iD}(t)|} \quad (8\text{-}8)$$

Let λ_{ij} be the rate of unwanted packets from node i to node j. We can also extend Eqn. 6-8 for the FCR for all nodes in T time period as follows:

$$FCR = \int_0^T \frac{\sum_{i=1}^{n} \lambda_{ij} \times FCR_{ij}(t) \times |F_{ij}(t)|}{\sum_{i=1}^{n} \lambda_{ij} \times |F_{ij}(t)|} dt$$

8.10 Optimal Cases and Theoretical Bounds

We next compute the theoretical bound for maximum possible PDR and minimum FCR. Descriptions for the symbols and notation used here are in Table 8-2.

Parameters description	
N:Number of rules	α: Zipf parameter
p_i: popularity of rule i	c: size of IPT
δ_ℓ: set of nodes in distance ℓ	δ^*: set of nodes farther than
λ_i:rate of unwanted packet generated by node i to destination	

Table 8-2: Parameter used for computing bounds

8.10.1 Maximum Packet Discarding Ratio

To maximize PDR for destination node D, we need to place rules so that the number of unwanted packets reaching the destination D is minimized. Thus, we want to put the maximum number of unique rules with hit rate as high as possible in the path from the unwanted packets source to the destination D. Let c be the IPT size. The best placement strategy of the rules that maximize PDR is as the following: (1) We sort the rules based on their hit rate from high to low. (2) We place $r_1, \ldots, r_c$ in 1-hop distance

nodes, $r_{c+1}, \dots, r_{2c}$ in 2-hop distance nodes, and accordingly $r_{(k-1)c+1}, \dots, r_{kc}$ in k-hop distance nodes (note that $k \leq N/c$). Thus, we put the first c rules with highest hit rates in all nodes within 1-hop distance so that they are effective in all possible paths to the node D. Similarly, we put the second c highest hit rate rules in all nodes within 2-hop distance and so forth. Clearly, for nodes with distance more than N/c, all unwanted packets are discarded before reaching the destination because all rules have already been placed in their path to the destination. Let δ_i denote the set of nodes i hops away from the destination and δ^* denote the set of nodes that are located farther than N/c hops distance. The total number of packets discarded before reaching the destination is denoted by μ_{max} and is calculated as follows:

$$\mu_{max} = \sum_{i=1}^{N/c} \sum_{j \in \delta_i} \lambda_j \sum_{k=1}^{ic} p_k + \sum_{j \in \delta^*} \lambda_j \quad (8\text{-}9)$$

Note that in Eqn. 6-9, the first term is the number of discarded packet generated by nodes within N/c hops from the destination. The second term is the number of discarded packets generated by nodes whose distance is farther than N/c to the destination. Replacing $\sum_{i=a}^{b} p_i$ by $\left(\frac{b^{1-\alpha}-a^{1-\alpha}}{1-\alpha}\right)\Omega^{-1}$ where $\Omega = \sum_{i=1}^{N} p_i = \frac{N^{1-\alpha}-1}{1-\alpha}$, Eqn. 6-9 can be written as:

$$\mu_{max} = \sum_{i=1}^{N/c} \sum_{j \in \delta_i} \lambda_j \frac{(ic)^{1-\alpha}-1}{N^{1-\alpha}-1} + \sum_{j \in \delta^*} \lambda_j \quad (8\text{-}10)$$

We can calculate the maximum packet discarding ratio as $PDR_{max} = \mu_{max}/(n \sum_{i=1}^{n} \lambda_i)$. Thus, using Eqn. 6-10, the upper-bound of PDR for node D is calculated as follows:

$$\overline{PDR_D} \leq \frac{\sum_{i=1}^{N/c} \sum_{j \in \delta_i} \lambda_j \frac{(ic)^{1-\alpha_{-1}}}{N^{1-\alpha_{-1}}} + \sum_{j \in \delta^*} \lambda_j}{n \sum_{i=1}^{n} \lambda_i} \quad (8\text{-}11)$$

8.10.2 Minimum Forwarding Cost Ratio

In order to minimize the forwarding cost ratio for a designated destination node D, we use the following bottom-up rule placement approach. All paths to a destination node can be represented by the shortest path spanning tree rooted at node D. To minimize FCR, we need to have the least number of packet forwardings which implies that each node needs to discard as many unwanted packets as possible. Note that rules are sorted based on their hit rate therefore the leaf nodes in the tree need to store rules from $r_1, \dots, r_c$ in order to discard the maximum number of unwanted packets. The next level of the nodes (*i.e.*leaves' parents) need to store the rules that match the maximum number of unwanted packets at the nodes. This includes the unwanted packets that the nodes generate locally and the unwanted packets they receive from their children. FCR is minimized if this process continues recursively along the spanning tree to the root. Since calculating the minimum value for FCR is difficult because of its dependency on the semantics of rules, network topology (*i.e.* number of children per node), node's capacity c and the skewness parameter of power law distribution α, we calculate a lower-bound for FCR as follows. Suppose node A is located within distance ℓ from the destination. To minimize FCR of unwanted packets generated by node A, we need to place rules such that rules $1, \dots, c$ are stored in node A, rules $c+1, \dots, 2c$ are stored in the first node in the path, and subsequently rules $ic+1, \dots, (i+1)c$ are stored inthe i-th node in the path. Therefore, the total number of forwarding for unwanted packets generated by node A within distance ℓ of node D denoted by F_ℓ^A is calculated as follows:

$$F_{\ell}^{A} = \left(\left(1 - \sum_{j=1}^{c} p_j\right) + \left(1 - \sum_{j=1}^{2c} p_j\right) + \cdots + \left(1 - \sum_{j=1}^{d\times c} p_j\right)\right) \times \lambda_A$$

$$= (\ell - \textstyle\sum_{j=1}^{\ell\times c} p_j) \times \lambda_A \quad (8\text{-}12)$$

However, as there are multiple sources of unwanted packets and each node has limited capacity of c, we cannot place the rules as described. For instance, let node B be the first node in the path from node A to the destination node. If node B also is a source of unwanted packets, it needs to store rules from $1, \ldots, c$, but its IPT is already full by rules $c + 1, \ldots, 2c$ that are imported in the first round for unwanted packets generated by node A. Hence, assuming Eqn. 6-12 holds for all nodes, which may not be applied in reality, we can calculate a lower-bound of FCR for destination node D as follows:

$$\overline{FCR_D} \geq \frac{\sum_{\ell=1} \sum_{\forall i \in \delta_\ell} F_d^i}{\sum_{\ell=1} \sum_{\forall i \in \delta_\ell} (\lambda_i \times \ell)} \quad (8\text{-}13)$$

Using Eqns. (6-13) and (6-14), the lower-bound for FCR can be stated as follows:

$$\overline{FCR_D} \geq 1 - \frac{\sum_{\ell=1} \sum_{\forall i \in \delta_\ell} (\sum_{j=1}^{\ell\times c} p_j \times \lambda_i)}{\sum_{\ell=1} \sum_{\forall i \in \delta_\ell} (\lambda_i \times \ell)} \quad (8\text{-}14)$$

8.11 Rule Admission and Replacement Policy

In this section, we present two rule admission and replacement policies: *Split Replacement Policy (SRP)* and *Proximity Aware Replacement (PAR*).

8.11.1 Split Replacement Policy Algorithm

As mentioned before, due to limited size of IPTs, PDR and FCR are dependent on the number of rule duplications in the paths to the destination. Thus, we should carefully

control the amount of rule duplication to tradeoff between the two potentially conflicting goals of minimizing FCR and maximizing PDR. To this end, we divide each IPT into two segments: the first segment of an IPT holds rules with highest hit rates with no constraints on number of duplications in the path. The second segment of an IPT stores the rules that are unique in the path. Keeping the rules with the highest hit rates in the first segment of an IPT helps decreasing FCR, and keeping the uniqueness of rules along the path to the destination in the second segment of IPT helps increasing PDR. By adjusting the size of the first and the second segments of IPTs, SRP can easily regulate the total cost of unwanted packets in the network.

Using SRP, when a node receives a new rule, it imports the rule, if it has a free slot in its IPT; otherwise, it executes the following procedure: (1) If the rule has been stored in the path ($d \neq 0$) and if there is a rule in the *first* segment of an IPT whose hit rate is smaller than the new rule's hit rate, the rule is replaced with the rule with the lowest hit rate; otherwise, the rule will not be imported in IPT. (2) If the rule has not been stored in the path ($d = 0$) and if there is a rule in the *second* segment of an IPT whose hit rate is smaller than the new rule's hit rate, the rule is replaced with the rule with the lowest hit rate; otherwise, the rule will not be imported in the IPT. In case a rule is added to an IPT, the rule admission distance is updated to node's distance to the destination and the rule will be forwarded to the next hop. SRP pseudo code is shown in Algorithm 1.1.

Algorithm 1: Split Replacement Policy

Input: rule r,rule hit rate h, rule admision distance d
node IPT :IPT
distance to the destination: ℓ
IPT size: c
Output: new IPT: IPT

if $(IPT.size \leqslant c)$ **then**
    $IPT.add(r)$
    $d \leftarrow \ell$
    return
else
    if $(\ d = 0)$ **then**
        $\hat{r} \leftarrow$ rule with lowest hit rate $\hat{h}$ in the second part of IPT
    else
        $\hat{r} \leftarrow$ rule with lowest hit rate $\hat{h}$ in the first part of IPT
    if $(h > \hat{h})$ **then**
        replace $\hat{r}$ with r
        $d \leftarrow \ell$
forward the rule to the next node in the path

8.11.2 Proximity Aware Replacement Algorithm

An alternate approach is to use the distance parameter to control the amount of rule duplication in a path. In PAR algorithm, a node accepts rule r_i if d_i is larger than a predefined threshold σ. Intuitively, when σ is small, high hit rate rules are stored in many nodes in the path which decreases both FCR and PDR. On the other hand, when σ is large, high hit rate rules are stored in few nodes in the path which increases both FCR and PDR. In PAR algorithm, a new rule is admitted if a rule is not stored in any node within distance σ. In case the node's IPT is not full, the node imports the new rule; otherwise, if there is a rule in the IPT whose hit rate is smaller that the new rule's hit rate, the rule is replaced with the rule with the lowest hit rate. If the new rule's hit rate is smaller than the

lowest hit rate in the IPT, the rule will not be imported in the IPT. PAR pseudocode is in Algorithm 2.

Algorithm 2: Proximity Aware Replacement

Input: rule r,rule hit rate h, rule admission distance d
node IPT :IPT
distance to the destination: ℓ
proximity threshold: σ
IPT size: c
Output: new IPT: IPT

if $(\ell - d \geqslant \sigma)$ **then**
  if $(IPT.size \leqslant c)$ **then**
    $IPT.add(r)$ accept and store new rule $d \leftarrow \ell$
  else
    $\hat{r} \leftarrow$ rule with lowest hit rate $\hat{h}$ in IPT
    if $(h > \hat{h})$ **then**
      replace $\hat{r}$ with r
      $d \leftarrow \ell$
forward the rule to the next node in the path

8.12 **Simulation Results**

We evaluate PDR and FCR based on our replacement and admission policies, namely SRP and PAR.

α	Zipf parameter	[0.5 … 1]
c	IPT size	[10 … 200]
θ	SRP parameter	[0 … 1]
σ	Proximity parameter	[0 … 10]
v	Movement speed	0 m/s (static), 1m/s (slow) and
N	Total number of rules	10000
n	Total number of nodes	100
A	Area	500 m x 2500 m
Simulation runs for 15000 seconds. Packets are generated every 2		

Table 8-3: Simulation parameters

8.12.1 SRP Performance

Recall that we divide IPT tables into two segments where the first segment stores high hit rate rules and the second segment stores unique rules in the path that rules are distributed. In practice, the high hit rate rules are duplicated in the first segment of the IPT for all nodes in the path. We used parameter θ to indicate the percentage of IPT reserved in the first segment. Figure 8-6(a) shows the impacts of θ on PDR. The results correspond to $\alpha = 0.8$ and IPT size of 100, which is equal to 1% of the total number of rules in the destination node. For small θ values, nodes only store rules which have not been stored by other intermediates nodes in the path. Hence, the total number of different rules stored in the path increases which in turn causes more number of unwanted packets to be discarded in the path. This explains why PDR is high for small values of θ.

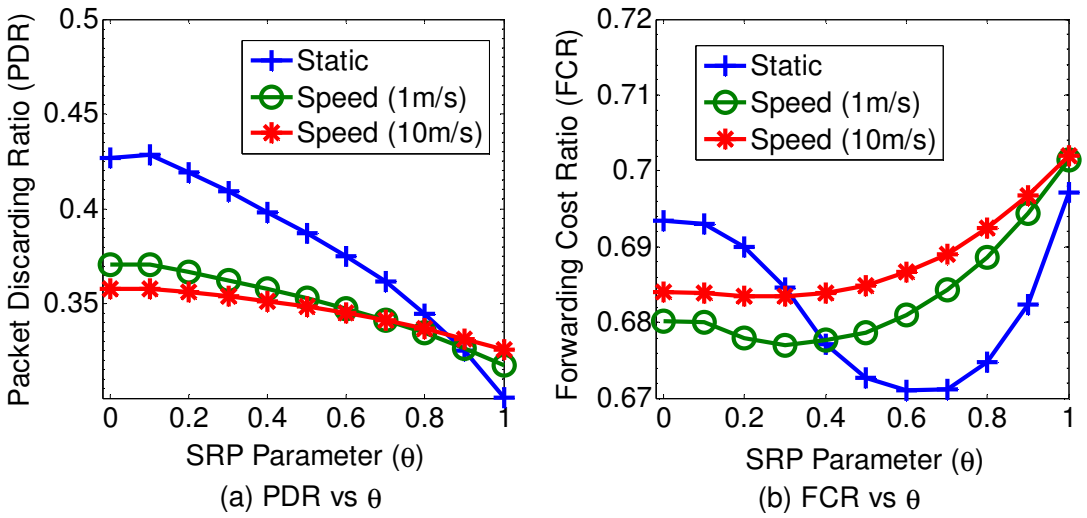


Figure 8-6: SRP Packet Discarding Ratio and Forwarding Cost Ratio for different θ values

However, as θ increases, the amount of rule duplication along the path that the rule is distributed increases and therefore PDR reduces. In an extreme case, when $\theta = 1$ (*i.e.*no second segment in IPT), all nodes will store the same set of high hit rate rules,

which in turn leads to the minimum PDR. The slope of decreasing trend of PDR is different for three mobility patterns. The more mobile a network is, the less significant the impact of θ is on PDR. The reason is when nodes are moving, rules in the second segment of IPT may not be unique along the path. Thus, the number of duplicated rules in a path stored in the second segment of IPT increases as nodes move faster. This unnecessary duplication of the rules in the second segment reduces the impact of θ on PDR.

Figure 8-6(b) shows the impact of θ on FCR. By increasing θ, greater number of high hit rate rules are stored in IPT. This leads to discarding more unwanted packets locally with minimum FCR of 0. However, as shown in Figure 8-6(a), by increasing θ, the number of packets that reaches the destination increases (*i.e.*PDR decreases). Thus, there is more number of unwanted packets with the maximum FCR of 1. The tradeoff between percentage of unwanted packets being discarded at the source and the percentage of unwanted packets that reach the destination brings up an optimal point for θ at which the average FCR is minimum. For instance, for a static network the θ that minimize FCR is 0.62. The optimal point for mobile networks shifts to the left as their mobility speed increases due to unnecessary rule duplications for the rules that are stored in the second segment of the IPT.

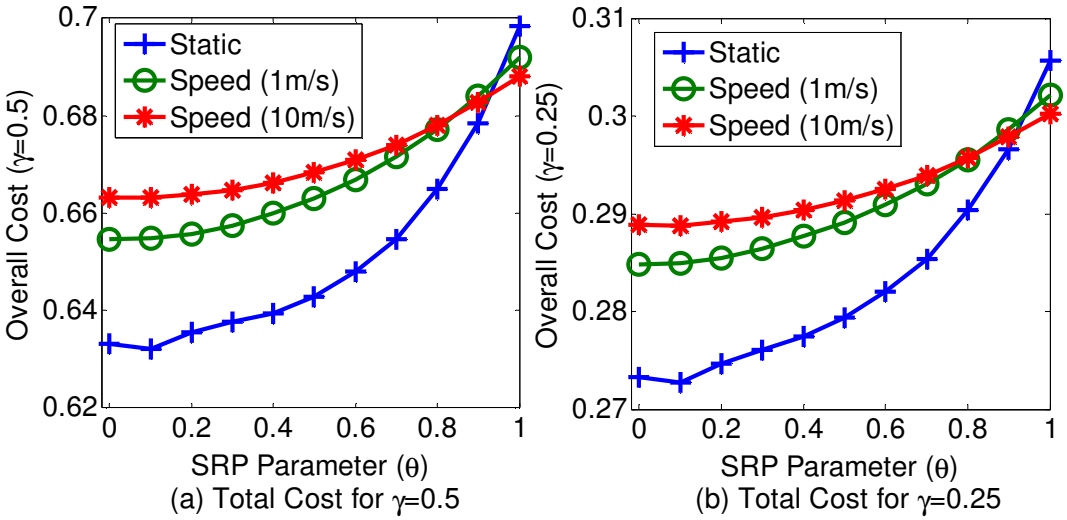


Figure 8-7: SRP overall cost

Figure 8-7(a) shows the impact of θ on the overall cost when $\gamma = 0.5$ (*i.e.*performance metrics PDR and FCR are equally important). In this case, the overall cost is minimum when $\theta = 0$ for mobile networks and $\theta = 0.1$ for static network. The results show in general the collaborative firewalling is indeed required to minimize the overall cost of unwanted packets. As in wireless mobile networks, FCR seems to be more important than PDR due to the power constraints on mobile nodes, Figure 8-7(b) shows the overall cost when $\gamma = 0.25$ where FCR is three times more important than PDR. The results show that the overall cost for SRP is down to 0.272 for static and 0.285 for mobile networks, respectively.

8.12.2 PAR Performance

In PAR, a node imports a rule only when its distance to the last node that stores the rule is less than a threshold σ. Figure 8-8(a) and (b) show the impact of σ on PDR and FCR, respectively. These graphs correspond to $\alpha = 0.8$ and IPT size of 100, which is equal to 1% of the total number of rules. In PAR, σ is inversely proportional to the

number of rule duplications along the path that the rule is distributed. Thus, increasing σ results in reducing rule duplications and therefore increasing PDR. Note that the PDR value is less for mobile network because of unnecessary rule duplications in a path. As an extreme case, when $\sigma = 0$, all nodes have the same set of rules in IPT (which is similar to SRP when $\theta = 1$).

Figure 8-8(b) shows the impact of σ on FCR. Similar to SRP, the number of duplicated rules creates a tradeoff between number of packets that are discarded at the source (*i.e.*FCR=0), and number of packets that are discarded at the destination (*i.e.*FCR=1). As number of duplicated rules in a path is determined by σ, there is an optimal σ at which FCR is minimum. For instance, the optimal σ for static network is equal to 1 and for mobile networks it varies from 2 to 3.

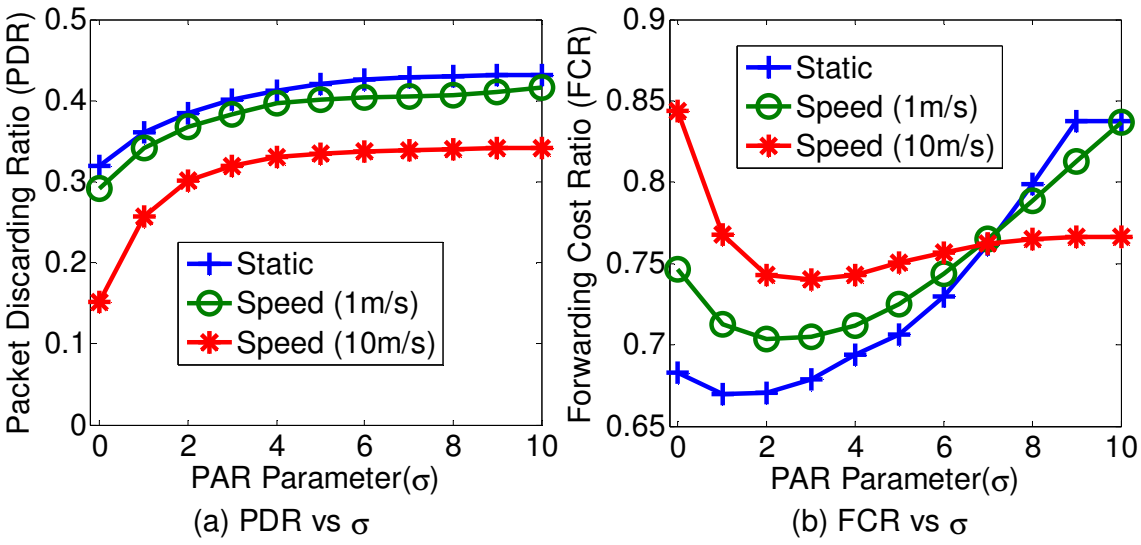


Figure 8-8: PAR Packet Discarding Ratio and Forwarding Cost Ratio for different σ values

Figure 8-9 (a) demonstrates the impacts of σ on the overall cost when $\gamma = 0.5$. The overall cost is minimum when $\sigma = 3$ for static network, and it varies from 4 to 4.2 for

mobile networks. Figure 8-9 (b) shows the overall cost when $\gamma = 0.25$. The results show that the overall cost for PAR is down to 0.28 for static and 0.29 for mobile network.

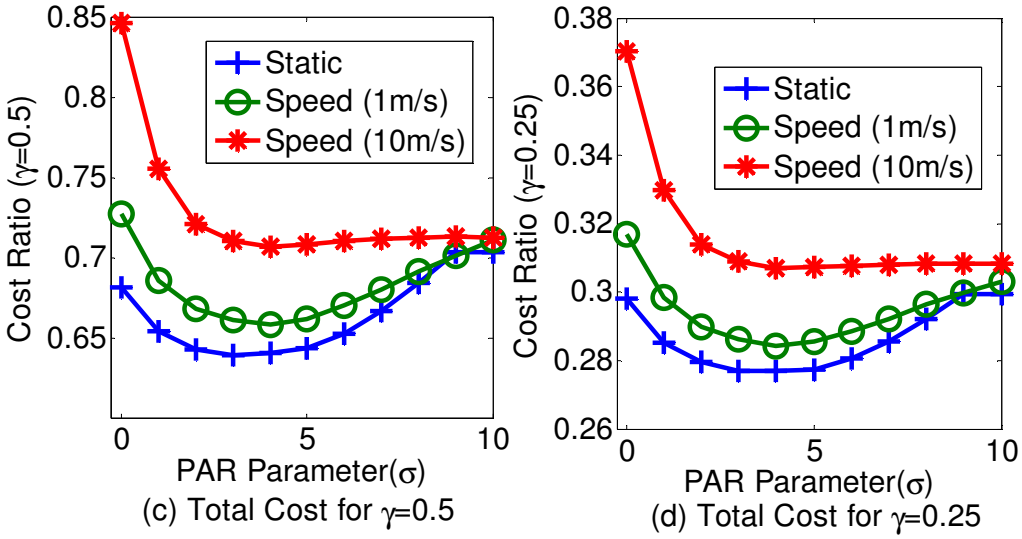


Figure 8-9: PAR overall cost

8.12.3 Sensitivity Analysis

Impacts of Mobility: We analyze the performance of the proposed scheme under three different mobility profiles: static network, random network mobility with 1m/s speed, and random network mobility with 10m/s speed. These mobility profiles are designed to represent wireless mesh networks, social mobile ad hoc networks, and vehicular ad hoc networks, respectively.

Wireless mesh networks usually contain static wireless routers, and they are modeled by the static network topology. As network and therefore paths are both static, rule duplication is controlled very well by PAR and SRP schemes in a static network. Maintaining the desired level of rule duplication results in high PDR and low FCR for static networks. Figure 8-10(a) and (b) show that the PDR and FCR are better using SRP

with $\theta < 0.83$ and $\theta > 0.39$, respectively. And Figure 8-8(a) and (b) show that the PDR and FCR are better using PAR with all σ values and $\sigma < 7$, respectively.

Social mobile ad-hoc network can be formed by collection of mobile devices carried by people who get together in university campuses, shopping malls, and other public places. To model such networks, we use random mobility with low speed of 1m/s. As the network topology changes gradually, rule duplications along the paths cannot be controlled as in static networks. Therefore, SRP and PAR are less effective which is observable from PDR and FCR graphs in Figure 8-6(a) and (b) and Figure 8-8(a) and (b).

Finally, we look at SRP and PAR performance for vehicular ad-hoc networks that are modeled by high-speed mobile networks with speed of 10m/s. When nodes move faster, the network topology become more dynamic. In other words, network paths changes more frequently over time, which in turn leads to less control on rule duplications along the paths. Hence, PDR decreases and FCR increases as nodes move faster.

Furthermore, by comparing the results from Figure 8-7 and Figure 8-9, PAR is more sensitive to mobility comparing to SRP, as the maximum cost difference between static and high speed mobile networks is 3 and 17 for SRP and PAR, respectively. In addition, for large values of θ and σ, the overall cost for different mobility profiles converge, which indicates that the number of rule duplications in the paths for both SRP and PAR replacement policy is almost the same.

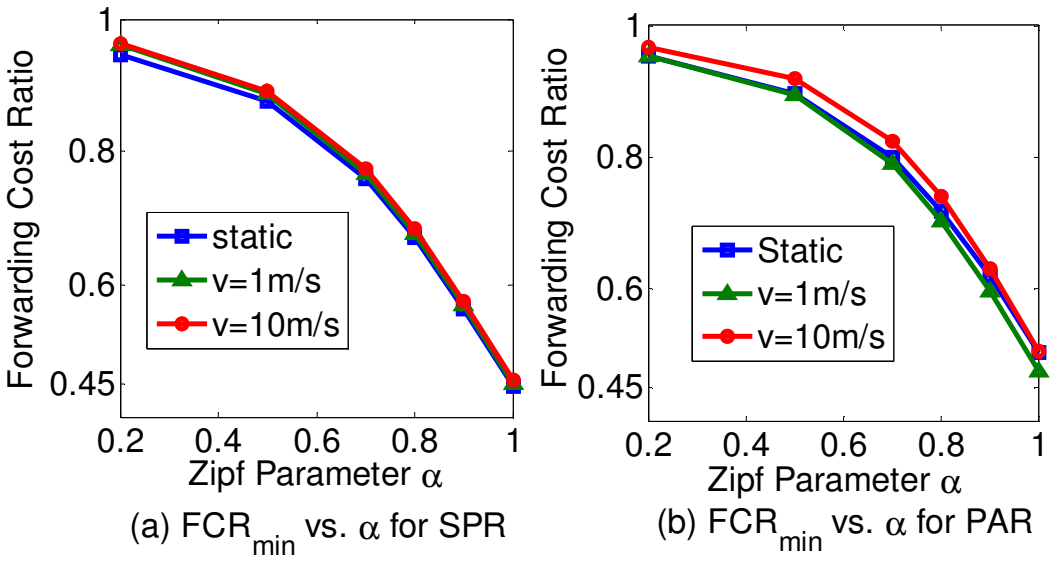


Figure 8-10: Impact of Zipf parameter (α) on FCR_{min}

Impacts of Zipf parameter α: The parameter α in Zipf distribution determines the skewness of the rule popularity distribution. In other words, α parameter is the slope of popularity function in log-log scale. Clearly, a Zipf distribution with $\alpha = 0$ represents a rule popularity with uniform distribution.

Figure 8-10 (a) and (b) show the impact of α on the minimum forwarding cost ratio FCR_{min} for SRP and PAR, respectively. To obtain the minimum FCR we use optimum value of θ and σ for SRP and PAR, respectively. Figure 8-10(a) and (b) illustrate that by increasing α, FCR reduces for both replacement algorithms. The main reason is that by increasing α less number of rules represents greater number of unwanted packets. Thus, the amount of traffic that is discarded at the source is higher which results in the decrease of the average FCR.

Figure 8-11 (a) and (b) show the maximum PDR in the network. The results illustrates that larger α values increases the percentage of the packets that are discarded

in a path. This is because a less number of rules match against a greater percentage of unwanted packets.

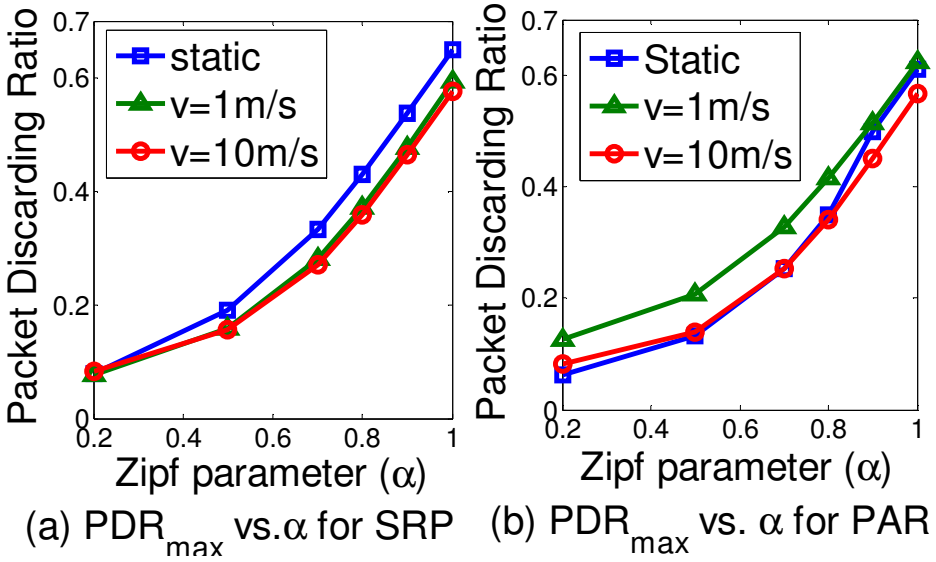


Figure 8-11: Impact of Zipf parameter (α) on PDR_{max}

Impacts of IPT size: Intuitively, FCR reduces if we store more number of the rules in each node. Thus, we expect a decreasing trend for FCR, while IPT size increases. Figure 8-12(a) and (b) show minimum FCR for SRP and PAR replacement algorithms, respectively. Interestingly, the impact of increasing IPT for different mobility speeds is different for SRP and PAR. It seems that the impact of increasing IPT in SRP is more consistent than PAR while nodes are moving fast.

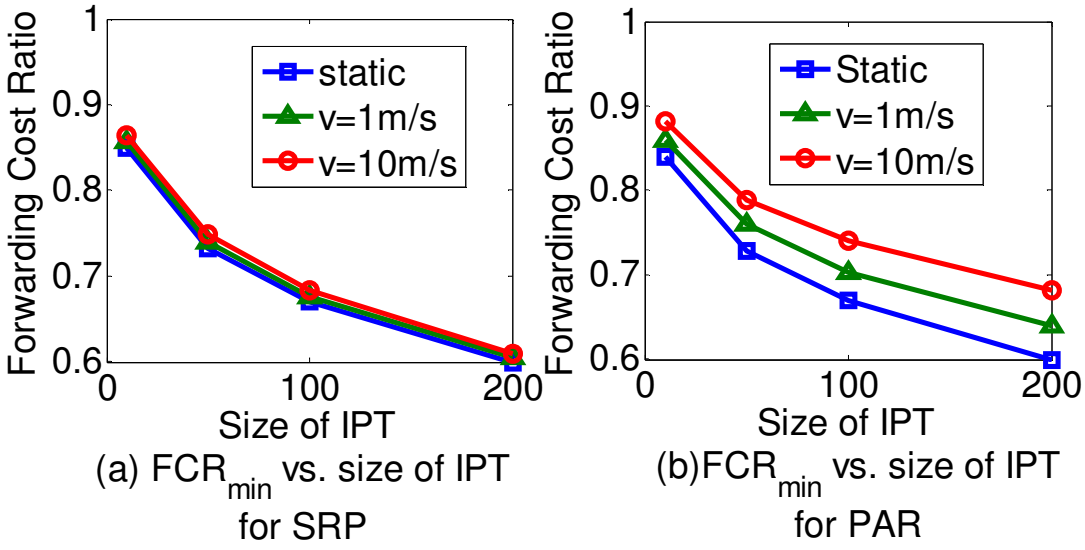


Figure 8-12: Impact of size of IPT on FCR_{min}

Similarly, Figure 8-13(a) and (b) demonstrate the impact of number of IPT size on maximum PDR. Indeed, by storing greater number of rules in each node, more unwanted packets are discarded along a path. This is the reason for increasing trend of PDR_{max} in both SRP and PAR.

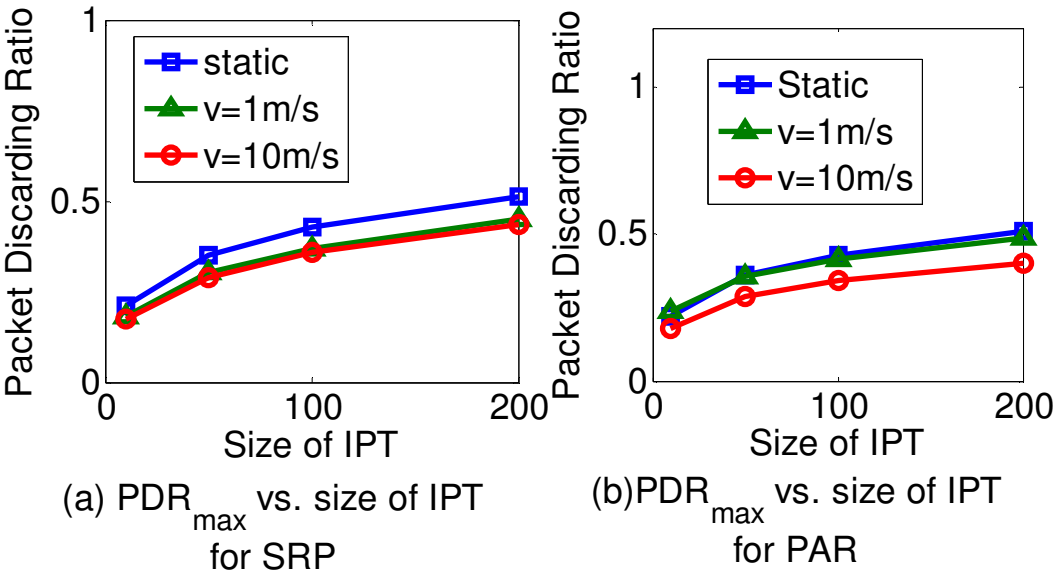


Figure 8-13: Impact of size of IPT on PDR_{max}

8.13 Summary and Conclusion

We proposed a collaborative firewalling scheme for mobile networks. We introduced two performance metrics, namely Packet Discarding Ratio (PDR) and Forwarding Cost Ratio (FCR), to address the effectiveness of distributed firewalling in the network. In addition, we developed an analytical model for the scheme using different system parameters and calculate theoretical bounds for system performance metrics. We further proposed two heuristic algorithms, namely the Proximity Aware Replacement and the Split Replacement Policy to maximize PDR and minimize FCR. We finally evaluated the performance of the system by extensive simulation on mobile networks with different mobility profiles. Our results show that using the proposed collaborative firewalling scheme, a considerable portion of unwanted traffic can be discarded before reaching the destinations, which saves substantial amount of power and bandwidth.

The simulation results revealed that for each node by distributing only 1% of its firewall rules, about 36% of unwanted packets can be discarded in mobile networks and about 42% in static networks. Furthermore, using *Split Cache* replacement policies, we can save the network bandwidth wasted by unwanted traffic up to 30% for different mobility patterns and speeds.

Chapter 9 : SUMMARY AND CONCLUSION

In this thesis we showed that cooperative content caching can effectively reduce the object provisioning cost for data enabled mobile devices. We proposed different caching strategies for a wide range of scenarios such as homogenous and heterogeneous networks. We also analyzed the performance of cooperative caching under various mobility patterns in social wireless networks.

In Chapter 3 we developed an optimal cooperative caching strategy that minimizes the object provisioning cost in a stationary network where all users have the same request generation pattern. We also developed an analytical model to compute the optimal split parameter to minimize the provisioning cost.

In Chapter 4 we extended the optimal strategy and developed a benefit based caching strategy to minimize the provisioning cost when users have different request generation patterns. In a heterogeneous network, users have different interest and they also have different request generation rate. By modeling the problem as a classic maximum weight matching in bipartite graphs we showed the upper bound performance for the cooperative caching. Our heuristic benefit based caching is able to provide a reasonable and comparable performance compared to the upper bound.

Performance of cooperative caching is highly dependent on node mobility patterns. In Chapter 5 we investigated the following mobility patterns and their impacts on performance of cooperative caching. a) *Random Walk*: Random walk is a simple and popular mobility pattern in which each node pauses in a waypoint for a while and then it selects a random destination before moving to the destination at a randomly selected speed. After reaching the destination, it again pauses, and then, repeats the above

behavior. b) *Human Walk*: we studied the performance of cooperative caching in social wireless networks based on real human mobility traces.

In Chapter 6 we analyzed the impacts of such selfish user behavior on the object provisioning cost and on the earned rebate in a social wireless network (*MSWNET*). In particular, we compared the provisioning cost under presence of selfish nodes with the optimal provisioning cost when all nodes are cooperative.

In Chapter 7 and Chapter 8 we introduced two applications of our cooperative caching scheme for improving object availability and also for discarding unwanted packets in mobile wireless networks.

9.1 Future Work

In Chapter 4 we proposed a heuristic for reducing the object provisioning cost in heterogeneous networks. It will be desirable to develop and implement a distributed maximum weight matching algorithm to get the optimal solution for these networks.

In Chapter 5 we proposed a caching strategy for minimizing the provisioning cost in community based social wireless networks. To detect the community we applied the existence community detection algorithms. A more detailed study will be useful to understand the impacts of community detection algorithms and performance of caching.

We investigated the impacts of user selfishness in Chapter 6. However, we did not propose any solution for preventing selfishness in the network. The possible solution would be detecting and punishing the selfish nodes by revoking its earned rebates. A game theory framework can be developed for analyzing the detail dynamics of the game between selfish and non-selfish nodes.

Analysis of selfishness can be also extended to heterogeneous mobile networks where users are highly mobile, have different interest and also different request generation rates.

BIBLIOGRAPHY

Chapter 10 BIBLIOGRAPHY

[1] "Cisco predicts wireless-data explosion." [Online]. Available: http://news.cnet.com/8301-30686_3-10449758-266.html. [Accessed: 13-Apr-2011].

[2] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010–2015," 01-Feb-2011. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html. [Accessed: 13-Apr-2011].

[3] "The Economist, A special report on managing information: Data, data everywhere," Feb-2010. [Online]. Available: http://www.economist.com/node/15557443?story_id=E1_TVVVSQQP. [Accessed: 13-Apr-2011].

[4] Tom Kaneshige, "AT&T IPhone Users Irate at Idea of Usage-Based Pricing - PCWorld," Dec-2009. [Online]. Available: http://www.pcworld.com/article/184589/atandt_iphone_users_irate_at_idea_of_usagebased_pricing.html. [Accessed: 13-Apr-2011].

[5] Marguerite Reardon, "Verizon: Usage Based Pricing." [Online]. Available: http://www.dailywireless.org/2011/03/01/verizon-usage-based-pricing/. [Accessed: 13-Apr-2011].

[6] X. Bao, U. Lee, I. Rimac, and R. R. Choudhury, "DataSpotting: offloading cellular traffic via managed device-to-device data transfer at data spots," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 14, pp. 37–39, Dec. 2010.

[7] B. Han, P. Hui, and A. Srinivasan, "Mobile data offloading in metropolitan area networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 14, pp. 28–30, Nov. 2010.

[8] K. Lee, I. Rhee, J. Lee, S. Chong, and Y. Yi, "Mobile data offloading: how much can WiFi deliver?," in *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, New York, NY, USA, 2010, pp. 26:1–26:12.

[9] B. Han, P. Hui, V. S. A. Kumar, M. V. Marathe, G. Pei, and A. Srinivasan, "Cellular traffic offloading through opportunistic communications: a case study," in *Proceedings of the 5th ACM workshop on Challenged networks*, New York, NY, USA, 2010, pp. 31–38.

[10] M. Valerio Barbera, J. Stefa, A. Carneiro Viana, M. Dias De Amorin, and M. Boc, "VIP Delegation: Enabling VIPs to Offload Data in Wireless Social Mobile Networks," INRIA, Research Report RR-7563, 2011.

[11] "T-Mobile @Home - General Information." [Online]. Available: http://support.t-mobile.com/doc/tm23449.xml#1. [Accessed: 13-Apr-2011].

[12] P. J. Denning, "The locality principle," *Commun. ACM*, vol. 48, pp. 19–24, Jul. 2005.

[13] George K. Zipf, *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[14] S. Jin and A. Bestavros, "Temporal Locality in Web Request Streams: Sources, Characteristics, and Caching Implications (Extended Abstract)," in *In Proceedings of SIGMETRICS*, 2000.

[15] A. Mahanti, D. Eager, and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance," *Performance Evaluation*, vol. 42, pp. 187–203, 2000.

[16] V. Almeida, A. Bestavros, M. Crovella, and A. D. Oliveira, "Characterizing Reference Locality in the WWW," 1996, pp. 92–103.

[17] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW Client-based Traces," 1995.

[18] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," *Networking, IEEE/ACM Transactions on*, vol. 5, no. 6, pp. 835 –846, Dec. 1997.

[19] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *In INFOCOM*, 1999, pp. 126–134.

[20] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of YouTube network traffic at a campus network - Measurements, models, and implications," *Computer Networks*, vol. 53, no. 4, pp. 501 – 514, 2009.

[21] X. Cheng, C. Dale, and J. Liu, "Statistics and Social Network of YouTube Videos," in *16th International Workshop on Quality of Service (IWQoS)*, 2008, pp. 229 –238.

[22] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, 2007, pp. 15–28.

[23] Y. Amiel and F. Cowell, "Monotonicity, dominance and the Pareto principle," *Economics Letters*, vol. 45, no. 4, pp. 447 – 450, 1994.

[24] R. Lancellotti, B. Ciciani, and M. Colajanni, "A Scalable Architecture for Cooperative Web Caching," in *Proceedings of Workshop in Web Engineering, Networking*, 2002.

[25] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A Hierarchical Internet Object Cache," in *Proceedings of the USENIX technical conference*, 1995, pp. 153–163.

[26] D. Wessels and k claffy, "ICP and the Squid Web Cache," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 345–357, 1998.

[27] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," 1998.

[28] Y. Du, S. K. S. Gupta, and G. Varsamopoulos, "Improving on-demand data access efficiency in MANETs with cooperative caching," *Ad Hoc Networks*, vol. 7, no. 3, pp. 579 – 598, 2009.

[29] C.-Y. Chow, H. Va Leong, and A. T. S. Chan, "Group-Based Cooperative Cache Management for Mobile Clients in a Mobile Environment," in *Proceedings of the 2004 International Conference on Parallel Processing (ICPP)*, Washington, DC, USA, 2004, pp. 83–90.

[30] C.-Y. Chow, H. V. Leong, and A. T. S. Chan, "GroCoca: group-based peer-to-peer cooperative caching in mobile environment," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 179 –191, Jan. 2007.

[31] K. Psounis and B. Prabhakar, "A randomized Web-cache replacement scheme," in *Proceedings Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2001, vol. 3, pp. 1407 –1415 vol.3.

[32] A. I. Vakali, "LRU-based algorithms for Web Cache Replacement," in *First Internat. Conf. on Electronic Commerce and Web Technologies, Lecture Notes in Computer Science*, 2000, pp. 409–418.

[33] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 158–170, 1998.

[34] D. Barbará and T. Imieliński, "Sleepers and workaholics: caching strategies in mobile environments," in *Proceedings of the International Conference on Management of data (SIGMOD)*, New York, NY, USA, 1994, pp. 1–12.

[35] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, pp. 1251–1265, 2003.

[36] S. K. . Gupta and P. K. Srimani, "A strategy to manage cache consistency in a disconnected distributed environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 7, pp. 686 –700, Jul. 2001.

[37] K.-L. Tan, J. Cai, and B. C. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 789–807, 2001.

[38] N. Dimokas, D. Katsaros, and Y. Manolopoulos, "Cache consistency in Wireless Multimedia Sensor Networks," *Ad Hoc Networks*, vol. 8, no. 2, pp. 214 – 240, 2010.

[39] J. Cao, Y. Zhang, G. Cao, and L. Xie, "Data Consistency for Cooperative Caching in Mobile Environments," *Computer*, vol. 40, no. 4, pp. 60 –66, Apr. 2007.

[40] J. Cai and K. Tan, "Energy-efficient selective cache invalidation," *Wireless Networks*, vol. 5, pp. 489–502, 1999.

[41] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: An adaptive cache invalidation method in mobile client/server environments," *Mobile Networks and Applications*, vol. 2, pp. 115–127, 1997.

[42] Q. Hu and D. Lee, "Cache algorithms based on adaptive invalidation reports for mobile environments," *Cluster Computing*, vol. 1, pp. 39–50, 1998.

[43] B. Zheng, J. Xu, and D. L. Lee, "Cache invalidation and replacement strategies for location-dependent data in mobile environments," *IEEE Transactions on Computers*, vol. 51, no. 10, pp. 1141 – 1153, Oct. 2002.

[44] J. C.-H. Yuen, E. Chan, K.-Y. Lam, and H. W. Leung, "Cache invalidation scheme for mobile computing systems with real-time data," *SIGMOD Rec.*, vol. 29, pp. 34–39, Dec. 2000.

[45] S. Lim, W.-C. Lee, G. Cao, and C. R. Das, "Cache invalidation strategies for internet-based mobile ad hoc networks," *Computer Communications*, vol. 30, no. 8, pp. 1854 – 1869, 2007.

[46] B. Y. Chan, A. Si, and H. V. Leong, "Cache management for mobile databases: design and evaluation," in *Proceedings of 14th International Conference on Data Engineering*, 1998, pp. 54 –63.

[47] T. Hara and S. K. Madria, "Consistency Management among Replicas in Peer-to-Peer Mobile Ad Hoc Networks," in *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, Washington, DC, USA, 2005, pp. 3–12.

[48] J. Cao, Y. Zhang, L. Xie, and G. Cao, "Consistency of cooperative caching in mobile peer-to-peer systems over MANET," in *25th IEEE International Conference on Distributed Computing Systems Workshops*, 2005, pp. 573 – 579.

[49] C. C. F. Fong, J. C. S. Lui, and M. H. Wong, "Quantifying complexity and performance gains of distributed caching in a wireless network environment," *International Conference on Data Engineering*, p. 104, 1997.

[50] J. Xu, X. Tang, and D. L. Lee, "Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, pp. 474–488, 2003.

[51] Y. Sawai, M. Shinohara, A. Kanzaki, T. Hara, and S. Nishio, "Consistency Management among Replicas Using a Quorum System in Ad Hoc Networks," in *Proceedings of the 7th International Conference on Mobile Data Management*, Washington, DC, USA, 2006, p. 128–.

[52] L. Y. Cao and M. T. Özsu, "Evaluation of Strong Consistency Web Caching Techniques," *World Wide Web*, 2002.

[53] S. Banerjee and S. Karforma, "A prototype design for DRM based credit card transaction in E-commerce," *Ubiquity*, May 2008.

[54] P. Koster and W. Jonker, "Digital Rights Management," in *Multimedia Retrieval*, H. M. Blanken, H. E. Blok, L. Feng, and A. P. Vries, Eds. Springer Berlin Heidelberg, 2007, pp. 321–345.

[55] Q. Liu, R. Safavi-Naini, and N. P. Sheppard, "Digital rights management for content distribution," in *Proceedings of the ACSW Frontiers*, Darlinghurst, Australia, 2003, pp. 49–58.

[56] D. KUNDUR and K. KARTHIK, "Video fingerprinting and encryption principles for digital rights management," *Proceedings of the IEEE*, vol. 92, no. 6, pp. 918 – 932, Jun. 2004.

[57] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding, "Cool-Tether: energy efficient on-the-fly wifi hot-spots using mobile phones," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, New York, NY, USA, 2009, pp. 109–120.

[58] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, New York, NY, USA, 2009, pp. 280–293.

[59] K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong, "SLAW: A New Mobility Model for Human Walks," in *IEEE INFOCOM*, 2009, pp. 855 –863.

[60] C. Boldrini, M. Conti, and A. Passarella, "Users mobility models for opportunistic networks: the role of physical locations," in *IEEE Wireless Rural and Emergency Communications - WRECOM07*, 2007, pp. 1–6.

[61] N. Eagle, A. Pentland, and D. Lazer, "{Inferring Social Network Structure using Mobile Phone Data}," *PNAS*, 2007.

[62] M. McNett and G. M. Voelker, "Access and mobility of wireless PDA users," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 2, pp. 40–55, Apr. 2005.

[63] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, *CRAWDAD data set cambridge/haggle (v. 2009-05-29)*. 2009.

[64] M. Morin, "The finacial impact of attack traffic on broadband networks," *Annual Review of Broadband Communications*, vol. 1, pp. 11–14, 2005.

[65] T. P. Kelly, Y. M. Chan, S. Jamin, C. Sugih, J. Jeffrey, and J. K. MacKie-Mason, "Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value," in *In Fourth International Web Caching Workshop*, 1999.

[66] S. Jin and A. Bestavros, "GreedyDual* Web Caching Algorithm – Exploiting the Two Sources of Temporal Locality in Web Request Streams," in *In proceedings of the 5th international web caching and content delivery workshop*, 2000, pp. 174–183.

[67] S. Jin and A. Bestavros, "Popularity-Aware GreedyDual-Size Web Proxy Caching Algorithms," in *Proceedings of ICDCS*, 1999, pp. 254–261.

[68] S. C. Rhea and K. Liang, "Value-Based Web Caching," in *In Proc. of the 12th Int. World Wide Web Conference*, 2003, pp. 619–628.

[69] P. Triantafillou and I. Aekaterinides, "Web Proxy Cache Replacement: Do's, Don'ts, and Expectations," in *Proceedings of The 2nd IEEE International Symposium on Network Computing and Applications (NCA)*, 2003.

[70] A. D. Bradley and A. Bestavros, "Basis token consistency: supporting strong Web cache consistency," in *IEEE Global Telecommunications Conference(GLOBECOM)*, 2002, vol. 3, pp. 2225 – 2229 vol.3.

[71] A. S. Z. Belloum and L. O. Hertzberger, "Concurrent Evaluation of Web Cache Replacement and Coherence Strategies," presented at the SIMULATION, 2002.

[72] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar, "Engineering Web Cache Consistency," *ACM Transactions on Internet Technology*, vol. 2, p. 2002, 2002.

[73] M. Mikhailov, "Evaluating a New Approach to Strong Web Cache Consistency With Snapshots of collected content," in *In international World Wide Web Conference*, 2003, pp. 599–608.

[74] C. Liu and P. Cao, "Maintaining Strong Cache Consistency in the World-Wide Web," in *Proceedings of the Seventeenth International Conference on Distributed Computing Systems*, 1998, pp. 445–457.

[75] E. Cohen and H. Kaplan, "Refreshment Policies for Web Content Caches," in *In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2001, pp. 1398–1406.

[76] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," in *Proceedings of the USENIX Technical Conference*, 1996, pp. 141–151.

[77] J. Yang, W. Wang, and R. Muntz, "Collaborative Web Caching Based on Proxy Affinities," in *Proceedings of ACM SIGMETRICS*, 2000, pp. 78–89.

[78] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching," in *ACM Symposium on Operating Systems Principles*, 1999, pp. 16–31.

[79] C. M. Bowman, M. O. T. T. Staff, D. R. Hardy, P. B. Danzig, A. Professor, C. Dept, and U. S. California, "The Harvest Information Discovery and Access System," in *Computer Networks and ISDN Systems*, 1995, pp. 763–771.

[80] J. Xu, Q. Hu, W. Lee, and D. L. Lee, "Performance Evaluation of an Optimal Cache Replacement Policy for Wireless Data Dissemination," *IEEE Transaction of Knowledge and Data Eng.*, vol. 16, pp. 125–139, 2001.

[81] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 404–418, 2001.

[82] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Beyond hierarchies: Design considerations for distributed caching on the internet," Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS), 1998.

[83] C. Lindemann and O. P. Waldhorst, "Evaluating Cooperative Web Caching Protocols for Emerging Network Technologies," in *in: Proceedings of Workshop on Caching, Coherence and Consistency (WC3)*, 2001.

[84] Y. Zhu, "Exploiting client caches: An approach to building large web caches," in *In Proceedings of the International Conference on Parallel Processing (ICPP)*, 2002.

[85] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305 – 1314, Sep. 2002.

[86] Y. Mao, Z. Zhu, and W. Shi, "Peer-to-peer web caching: Hype or reality," in *Proceedings of the tenth International Conferences on Parallel and Distributed Systems*, 2004, pp. 171–178.

[87] M. Busari and C. Williamson, "Simulation evaluation of a heterogeneous Web proxy caching hierarchy," in *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT)*, 2001, pp. 379 –388.

[88] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a decentralized peer-to-peer web cache," in *Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC)*, New York, NY, USA, 2002, pp. 213–222.

[89] M. Taghizadeh, A. Plummer, and S. Biswas, "Cooperative caching for improving availability in Social Wireless Networks," in *IEEE 7th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2010, pp. 342 –351.

[90] F. Sailhan and V. Issarny, "Cooperative Caching in Ad Hoc Networks," in *Proceedings of the 4th International Conference on Mobile Data Management (MDM)*, 2003, pp. 13–28.

[91] M. F. Caetano, J. L. Bordim, and M. A. . Dantas, "A collaborative cache approach for mobile ad hoc networks," in *IEEE Symposium on Computers and Communications (ISCC)*, 2009, pp. 404 –410.

[92] N. Chauhan, L. K. Awasthi, N. Chand, R. C. Joshi, and M. Mishra, "A cooperative caching strategy in mobile ad hoc networks based on clusters," in *Proceedings of the International Conference on Communication, Computing and Security (ICCCS)*, New York, NY, USA, 2011, pp. 17–20.

[93] T. Hara, "Cooperative caching by mobile clients in push-based information systems," in *Proceedings of the eleventh international conference on Information and knowledge management (CIKM)*, New York, NY, USA, 2002, pp. 186–193.

[94] T. Hara, K. Maeda, Y. Ishi, W. Uchida, and S. Nishio, "Cooperative caching by clients constructing a peer-to-peer network for push-based broadcast," *Data Knowl. Eng.*, vol. 69, pp. 229–247, Feb. 2010.

[95] Y.-H. Wang, C.-F. Chao, S.-W. Lin, and W.-T. Chen, "A distributed data caching framework for mobile ad hoc networks," in *Proceedings of the international conference on Wireless communications and mobile computing (IWCMC)*, New York, NY, USA, 2006, pp. 1357–1362.

[96] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 1, pp. 77 – 89, Jan. 2006.

[97] J. Zhao, P. Zhang, and G. Cao, "On Cooperative Caching in Wireless P2P Networks," in *The 28th International Conference on Distributed Computing Systems (ICDCS)*, 2008, pp. 731 –739.

[98] C.-Y. Chow, H. V. Leong, and A. T. S. Chan, "Distributed group-based cooperative caching in a mobile broadcast environment," in *Proceedings of the 6th international conference on Mobile Data Management (MDM)*, New York, NY, USA, 2005, pp. 97–106.

[99] N. Dimokas, D. Katsaros, and Y. Manolopoulos, "Cooperative caching in wireless multimedia sensor networks," *Mob. Netw. Appl.*, vol. 13, pp. 337–356, Aug. 2008.

[100] J. Zhao, P. Zhang, G. Cao, and C. R. Das, "Cooperative Caching in Wireless P2P Networks: Design, Implementation, and Evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 229 –241, Feb. 2010.

[101] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," in *Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2004, vol. 4, pp. 2537 – 2547 vol.4.

[102] C.-Y. Chow, H. V. Leong, and A. Chan, "Cache Signatures for Peer-to-Peer Cooperative Caching in Mobile Environments," in *Proceedings of the 18th International Conference on Advanced Information Networking and Applications(AINA)*, Washington, DC, USA, 2004, p. 96–.

[103] C.-Y. Chow, H. V. Leong, and A. Chan, "Peer-to-Peer Cooperative Caching in Mobile Environments," in *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Washington, DC, USA, 2004, pp. 528–533.

[104] S. Lim, W.-C. Lee, G. Cao, and C. R. Das, "A novel caching scheme for improving Internet-based mobile ad hoc networks performance," *Ad Hoc Networks*, vol. 4, no. 2, pp. 225 – 239, 2006.

[105] B. Tang, H. Gupta, and S. R. Das, "Benefit-based data caching in ad hoc networks," in *Proceedings of the 14th IEEE International Conference on Network Protocols (ICNP)*, 2006, pp. 208–217.

[106] E. Chan, W. Li, and D. Chen, "Energy saving strategies for cooperative cache replacement in mobile ad hoc networks," *Pervasive and Mobile Computing*, vol. 5, no. 1, pp. 77 – 92, 2009.

[107] W. Li, E. Chan, and D. Chen, "Energy-Efficient Cache Replacement Policies for Cooperative Caching in Mobile Ad Hoc Network," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2007, pp. 3347 –3352.

[108] P. Nuggehalli, V. Srinivasan, and C.-F. Chiasserini, "Energy-efficient caching strategies in ad hoc wireless networks," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc)*, New York, NY, USA, 2003, pp. 25–34.

[109] L. Yin, G. Cao, and Y. Cai, "A Generalized Target-Driven Cache Replacement Policy for Mobile Environments," in *The International Symposium on Applications and the Internet*, 2003, pp. 14–21.

[110] T. P. Sharma, R. C. Joshi, and M. Misra, "Dual radio based cooperative caching for wireless sensor networks," in *16th IEEE International Conference on Networks (ICON)*, 2008, pp. 1 –7.

[111] C.-Y. Chow, H. V. Leong, and A. Chan, "Peer-to-peer cooperative caching in a hybrid data delivery environment," in *Proceedings. 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, 2004, pp. 79 – 84.

[112] J. Xu, Q. Hu, D. L. Lee, and W.-C. Lee, "SAIU: an efficient cache replacement policy for wireless on-demand broadcasts," in *Proceedings of the ninth international conference on Information and knowledge management*, New York, NY, USA, 2000, pp. 46–53.

[113] H. Artail, H. Safa, K. Mershad, Z. Abou-Atme, and N. Sulieman, "COACS: A Cooperative and Adaptive Caching System for MANETs," *IEEE Transactions on Mobile Computing*, vol. 7, pp. 961–977, 2008.

[114] B. Tang and H. Gupta, "Cache placement in sensor networks under update cost constraint," in *Proceedings of AdHoc-Now*, 2005.

[115] G.-M. Chiu and C.-R. Young, "Exploiting In-Zone Broadcasts for Cache Sharing in Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 8, pp. 384–397, 2009.

[116] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," in *Mobile Computing*, vol. 353, T. Imielinski and H. F. Korth, Eds. Springer US, 1996, pp. 331–361.

[117] H. Shen, M. Kumar, S. K. Das, and Z. Wang, "Energy-efficient data caching and prefetching for mobile devices based on utility," *Mob. Netw. Appl.*, vol. 10, pp. 475–486, Aug. 2005.

[118] K. Y. Lai, Z. Tari, and P. Bertok, "Location-aware cache replacement for mobile environments," in *Global Telecommunications Conference (GLOBECOM)*, 2004, vol. 6, pp. 3441 – 3447 Vol.6.

[119] K. Y. Lai, Z. Tari, and P. Bertok, "Mobility-Aware Cache Replacement for Users of Location-Dependent Services," *Annual IEEE Conference on Local Computer Networks*, pp. 50–58, 2004.

[120] A. Kumar, M. Misra, and A. K. Sarje, "A weighted cache replacement policy for location dependent data in mobile environments," in *Proceedings of ACM symposium on Applied computing (SAC)*, New York, NY, USA, 2007, pp. 920–924.

[121] Y. Wang, E. Chan, W. Li, and S. Lu, "Location Dependent Cooperative Caching in MANET," *International Conference on Parallel Processing*, vol. 0, pp. 470–477, 2008.

[122] Y. Huang, J. Cao, and B. Jin, "A predictive approach to achieving consistency in cooperative caching in MANET," in *Proceedings of the 1st international conference on Scalable information systems (InfoScale)*, New York, NY, USA, 2006.

[123] N. Laoutaris, G. Smaragdakis, A. Bestavros, and I. Stavrakakis, "Mistreatment in Distributed Caching Groups: Causes and Implications," in *Proceedings of 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006, pp. 1 –13.

[124] N. Laoutaris, G. Smaragdakis, A. Bestavros, I. Matta, and l Stavrakakis, "Distributed Selfish Caching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1361 –1376, Oct. 2007.

[125] G. Smaragdakis, N. Laoutaris, I. Matta, A. Bestavros, and I. Stavrakakis, "A Feedback Control Approach to Mitigating Mistreatment," in *Proceedings of IFIP Networking*, 2006.

[126] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, "Selfish caching in distributed systems: a game-theoretic analysis," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (PODC)*, New York, NY, USA, 2004, pp. 21–30.

[127] M. Goemans, L. E. Li, V. S. Mirrokni, and M. Thottan, "Market sharing games applied to content distribution in ad-hoc networks," in *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*, New York, NY, USA, 2004, pp. 55–66.

[128] T. Hara, "Replica Allocation in Ad Hoc Networks with Periodic Data Update," in *Proceedings of the Third International Conference on Mobile Data Management*, Washington, DC, USA, 2002, pp. 79–86.

[129] J. Huang and M. Chen, "On the effect of group mobility to data replication in ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 5, 2006.

[130] T. Hara, "Quantifying Impact of Mobility on Data Availability in Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 241–258, Feb. 2010.

[131] L. Maccari, R. Fantacci, P. Neira, and R. M. Gasca, "Mesh network firewalling with Bloom Filters," in *Proceedings ICC*, 2007.

[132] M. Alicherry and A. D. Keromytis, "DIPLOMA: Distributed Policy Enforcement Architecture for MANETs," in *Proceedings pf International Conference on Network and System Security (NSS)*, 2010.

[133] M. Alicherry, A. D. Keromytis, and A. Stavrou, "Deny-by-Default Distributed Security Policy Enforcement in Mobile Ad Hoc Networks," *Proceedings pf International ICST Conference on Security and Privacy in Communication Networks*, 2009.

[134] M. Zhao, L. Mason, and W. Wang, "Empirical study on human mobility for mobile wireless networks," in *MILCOM*, 2008.

[135] *Cambridge trace file, Human interaction study*.

[136] S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, pp. 374–398, Dec. 2003.

[137] E. Cohen, B. Krishnamurthy, and J. Rexford, "Evaluating Server-Assisted Cache Replacement in the Web," in *Algorithms — ESA' 98*, vol. 1461, G. Bilardi, G. Italiano, A. Pietracaprina, and G. Pucci, Eds. Springer Berlin / Heidelberg, 1998, p. 1–1.

[138] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," *RFC3561*, 2003.

[139] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of Human Mobility on Opportunistic Forwarding Algorithms," *IEEE Computer Society*, no. IEEE Transactions on Mobile Computing, pp. 606–620, 2007.

[140] A. Schrijver, *Theory of Linear and Integer Programming*. Wiley, 1998.

[141] H. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, pp. 83–97, 1955.

[142] *Six Months of Web Client Traces of Boston University*. .

[143] "National Lab of Applied Network Research, Sanitized access log," Jul-1997. [Online]. Available: ftp://ircache.nlanr.net/Traces. [Accessed: 14-Apr-2011].

[144] "NASA Kennedy Space Center WWW server access log." [Online]. Available: ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz. [Accessed: 14-Apr-2011].

[145] "University of Saskatchewan's WWW server access log." [Online]. Available: ftp://ita.ee.lbl.gov/traces/usask_access_log.gz. [Accessed: 14-Apr-2011].

[146] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453, no. 7196, pp. 779–782, Jun. 2008.

[147] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket switched networks and human mobility in conference environments," in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, New York, NY, USA, 2005, pp. 244–251.

[148] M. Musolesi and C. Mascolo, "Designing Mobility Models based on Social Network Theory," *ACM SIGMOBILE Mobile Computing and Communication Review*, vol. 11, pp. 59–70, 2007.

[149] Kyunghan Lee, Seongik Hong, Seong Joon Kim, Injong Rhee, and Song Chong, "SLAW: A New Mobility Model for Human Walks," in *INFOCOM 2009, IEEE*, 2009, pp. 855–863.

[150] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.

[151] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002.

[152] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *PHYS.REV.E*, vol. 69, p. 066133, 2004.

[153] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[154] J. L. Rodgers and A. W. Nicewander, "Thirteen Ways to Look at the Correlation Coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[155] A. K. Ghosh, "Defense against Cyber Attacks on Mobile, Ad Hoc Network Systems (MANETs)," *BAA04-18. Proposer Information Pamphlet (PIP) DARPA Advanced Technology Office (ATO)*, 2004.

[156] M. G. Gouda and A. X. Liu, "Structured Firewall Design," *Computer Networks Journal (Elsevier)*, vol. 51, no. 4, pp. 1106–1120, Mar. 2007.

[157] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," *RFC 3626*, 2003.

[158] P. B. C. E. perkins, "Highly Dynamic Destination-Sequenced Distance Vector (DSDV) for Mobile Computers, Protocols and Applications," *Proceedings SIGCOMM*, 1994.

[159] Y. H. D. Johnson, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," *RFC4728*, 2007.

[160] P. Wang, D. S. Reeves, and P. Ning, "Secure Address Auto-configuration for Mobile Ad Hoc Networks," in *in Proceedings MOBIQUITOUS*, 2005.