THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# HYBRID DATA ACCESS IN SMARTPHONE-BASED WIRELESS NETWORKS

MAX FREILICH
SUMMER 2013

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Guohong Cao
Professor of Computer Science and Engineering
Thesis Supervisor

John Hannan
Associate Professor of Computer Science and Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

# ABSTRACT

The current generation of smartphones allows for the creation of novel applications that can take advantage of a variety of wireless communication interfaces. However, because many applications require internet connectivity, users must rely on cellular data networks using technologies like 3G. For some, 3G may be cost prohibitive or otherwise unusable. For those that do have 3G access, normal internet usage and web browsing may cause the 3G interface to incur significant delay and unnecessary energy consumption, degrading the user experience and shortening the battery life of the smartphone. To address these issues, nearby smartphones can be leveraged through ad hoc communication to work together to access data more energy and delay efficiently. Two hybrid data access schemes are proposed; in the first scheme, smartphones without 3G leverage cooperative cache-based ad hoc networking to request data from nearby smartphones that can access 3G. Not only can they access data without an active internet connection, but the data access delay is reduced through cooperative caching. In the second scheme, data requests of nearby 3G-enabled smartphones are forwarded to a single smartphone that performs the requests over its own 3G interface. By aggregating requests, this approach significantly reduces the average access delay and energy consumption among the smartphones in the cluster by reducing unnecessary tail energies and 3G network connection delays.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Professor Guohong Cao for his valuable advice and support in both thesis research and coursework. Additionally, I would like to thank my academic advisor Professor John Hannan for his guidance during my time at Penn State. Finally, I would like to thank my mother and father for their unconditional love throughout my life.

# Chapter 1

# Introduction

In recent years, the introduction and proliferation of mobile smartphones have created new levels of connectivity and significant opportunities for novel mobile computing applications. Applications that were once relegated to higher-powered devices like laptop and desktop computers can now seamlessly run on equally-powered, yet smaller and more mobile smartphone platforms. In addition to functioning as mobile telephones that are able to access the mobile 3G/4G networks for telephony, modern smartphones come equipped with powerful processors and storage, and specially-designed operating systems tuned for mobile performance, such as Android and iOS. In addition, most full-featured smartphones come equipped with a variety of sensors and communication interfaces that allow applications to fully take advantage of the phone's mobility.

The combination of processing power and a variety of hardware capabilities allow smartphones to be used for exciting and novel purposes, ranging from social networking to battlefield communication and management. This new technology has allowed researchers to design novel systems that, until recently, were not feasible. In [1], Lu et al. present a framework for sensing and classifying sound events, implemented for the iPhone. The system uses the built-in microphone to capture sound, and the sound data undergoes multi-stage processing and classification. The iPhone enables a novel application such as this, due to the API support for manipulating sensors like the microphone, and for the processing power necessary to perform this type of signal processing on-demand. In [2], Koukoumidis, Peh, and Martonosi develop an application that allows users running window-mounted iPhones to collaboratively predict traffic light schedules, in the aim of influencing driving patterns to decrease vehicle fuel consumption. The application takes advantage of the iPhone's built-in video camera for determining traffic light status,

and communication interfaces for collaborative sharing of traffic light schedules among nearby vehicles. In [3],Yan, Kumar, and Ganesan leverage the internet connectivity and camera hardware of the iPhone to implement a system that allows smartphone users to crowd source image recognition. The user is able to record a photo and, through the CrowdSearch service, request human workers through Amazon's Mechanical Turk program to identify the picture. This system demonstrates the potential for smartphones to gather data and offload work via internet connectivity.

However, the ability to take advantage of new hardware and software features creates some additional challenges. While processors and related hardware have gotten smaller and faster, energy management has become an increasingly important challenge. Today, most smartphones use Lithium-ion-based batteries, and as a result, battery capacity for smartphones has not improved significantly over the years. Because of this, battery capacity and energy consumption severely limit the capabilities of devices to run many processor or hardware-intensive operations [4]. While research like [5], [6], and [7] try to tackle the problem, application developers across platforms are not even given a standard means of determining the energy usage patterns and behaviors of their applications. As such, there have been methods and techniques devised to reduce energy consumption. In [8], energy-intensive code is offloaded to a remote server, processed, and sent back to the phone – with the goal to avoid performing some energy inefficient tasks on the phone. Still, there are very few standard ways for developers to adequately estimate and manage energy consumption.

Connectivity and communication delay represent additional challenges facing smartphones and application developers. Most smartphones come equipped with multiple communication interfaces that can take advantage of various pieces of communication infrastructure. The radio interface allows smartphones users to communicate via the cellular infrastructure (3G/4G) data networks. In locations with strong cellular infrastructure and little interference, cellular data networks can provide a stable and fast connection. However, in a location with sparse cellular base station coverage or significant radio interference, cellular data connections can be unreliable. This provides unique challenges for travelers and

other highly mobile smartphone users, for example. In addition, cellular data plans cost average users a significant amount of money for usually only a limited amount of network bandwidth.

For communication over 802.11 WiFi, many smartphones include a WiFi interface. Using WiFi, users can connect to infrastructure such as routers or wireless access points (APs) with generally greater connectivity and data rates than 3G. In many cases, WiFi access is offered as a free service. However, the range of wireless APs or routers is generally limited to tens or the low hundreds of meters and the highest concentration of APs is generally found in more urban, populated areas. Those looking to rely only on WiFi access for internet connectivity may have trouble staying connected as they travel or move from location to location.

For shorter range communication and file sharing, many smartphones also possess a Bluetooth interface. Using the Bluetooth protocol, users can pair with each other and exchange data over several meters [9]. However, this is not commonly used for internet connectivity; rather, it is used for simple file-sharing applications, for example. In addition to connectivity challenges, using the 3G/4G, WiFi, and Bluetooth interfaces also introduces additional challenges regarding energy consumption and properly managing the duration and effectiveness of the hardware interfaces.

While most smartphones have at least a few communication interfaces (3G, WiFi and Bluetooth) the ability to access data over the internet depends largely upon the network infrastructure available. Cellular data (for example, 3G) is in many cases nearly ubiquitous: in many urban and highly populated suburban areas, 3G coverage can be extremely strong. In traces collected in [10], for example, it was found that mobile phone users in Houston, Texas were covered by cellular networks 99% of the time. The percentage of the time that users had access to a WiFi AP was only 49%, and can be much lower depending upon the specific location. Similarly, many metropolitan and other highly populated areas have good cellular network 3G coverage, while in general users may only intermittently (depending on geography and mobility patterns) have access to a wireless AP. While 3G coverage may usually be available, cellular data plans remain expensive and lack flexibility. If a user cannot afford or otherwise

chooses not to purchase a 3G data plan, then the user must rely on WiFi coverage. However, WiFi AP coverage can be sporadic at best. In many situations, users without 3G access cannot access the internet. With today's smartphone applications becoming increasingly reliant on internet connectivity, a lack of internet connectivity can make a user's most critical applications useless.

Even when smartphone users have access to 3G coverage and possess the necessary data plan, there are still 3G usage challenges that users face – many times without being aware. These challenges stem from the way in which the 3G backbone network configures the 3G interface's energy state. If a smartphone user does not access cellular data frequently, then every time the user wishes to access the internet the device incurs a large 3G connection delay (up to 2 seconds), as well as unnecessary energy usage. This is known as the tail energy, and the management of this energy is frequently termed the "tail energy problem". For many users that browse the web, for example, they may spend a minute or several minutes before browsing to a new page. Every subsequent request can incur this two second delay and this wasteful tail energy. Research on the 3G interface has resulted in numerous proposals and protocols for mitigating the tail energy, but it still remains a challenge facing many smartphone users.

When smartphone users congregate in close proximity, ad hoc networking can be leveraged to create hybrid data access schemes to address the challenges described. In this thesis, two specific problems are addressed: the inability of smartphone users without 3G data plans to access data over the internet, and the delay and energy efficiency of requesting data over 3G. This thesis makes two contributions by addressing these problems in two different parts. In the first part, an ad hoc networking scheme using cooperative caching is designed and implemented to allow smartphone users without 3G access to request data from a nearby 3G-enabled smartphone, and to reduce their access delays with caching. In the second part, a scheme allowing smartphones in a cluster to offload their data requests to a designated smartphone, significantly reducing average energy consumption and data access delay, is proposed and evaluated through simulation.

The remainder of the thesis is organized as follows: in Chapter 2, the design and implementation of the cooperative-cached based data access scheme is presented and evaluated. The smartphone-based request aggregation scheme is proposed and evaluated through simulation in Chapter 3. Chapter 4 includes the conclusion and future work.

# Chapter 2

# Cooperative Cache for Smartphones

## 2.1 Background and Related Work

Suppose many smartphone users are in a group in close proximity, such as in a shopping mall or lecture hall. Imagine that the number of smartphone users with 3G data plans is relatively small, and the 3G users are spread out among the crowd. Furthermore, there are no open WiFi access points that can allow the non-3G users access to the internet. In this situation, the non-3G users have limited opportunities to access the internet. One option is for the non-3G users to befriend a 3G user and ask to request some data, and then transfer this data over a protocol such as Bluetooth. While this may be simple enough for 1 or 2 non-3G users, this solution can quickly become burdensome and unmanageable with many non-3G users. Another similar option is use tethering to gain connectivity from a 3G smartphone user through WiFi. While this may be an easier and more automated solution, it still requires manual steps on behalf of the smartphone users to set up. In addition, many smartphone carriers do not support tethering. Even if the carrier does allow some limited tethering, there are usually bandwidth restrictions and the number of simultaneous allowed devices can also be severely limited. Finally, because the 3G devices are located sporadically among the non-WiFi devices, some devices may not be within tether range to allow for tethering.

Instead, the smartphones can form an ad hoc network over WiFi. Using this network, non-3G smartphones can request data from the 3G smartphones. The requests can be delivered to the 3G smartphones via the ad hoc network, and the response can be returned to the requesting smartphones via the ad hoc network. If the network has large data access delays, they can be substantially reduced by using cooperative caching.

While it is most common for smartphones to participate in infrastructure-based IEEE 802.11 networks via a wireless AP or client-server communication via the cellular data network, a wireless ad hoc network does not rely on an existing infrastructure to facilitate communication. In an IEEE 802.11 ad hoc network, the connected devices (nodes) of the network dynamically create the network infrastructure by acting as routers. Instead of relying on fixed infrastructure to forward packets, nodes participating in the network forward data amongst themselves, either to other nodes or to internet-connected devices. In this way, nodes rely on their neighboring nodes for requesting remote data. In an ad hoc network, nodes have a shared responsibility for ensuring that other nodes are properly serviced; if one node does not forward data, for example, this could create a network bottleneck and decrease the connectivity for the other nodes in the network. Therefore, one of the principle research topics for ad hoc networks is routing. Specialized routing protocols have to be developed for ad hoc networks. Unlike infrastructure-based networks, such as the internet, ad hoc network devices are not devices dedicated to routing, so routing protocols for ad hoc networks have to be designed for different network topology and condition requirements. Some of the popular routing protocols for ad hoc networks include Ad hoc On-Demand Distance Vector Routing (AODV) and Dynamic Source Routing (DSR). In contrast to other routing protocols popular in the internet, AODV and DSR are reactive protocols – they only find routes upon request, versus proactively finding routes before they are needed.

Ad hoc networks receive significant research attention because they allow network communication in unique and challenged environments, and provide some distinct benefits over infrastructure-based network communication. For one, ad hoc networks do not require time consuming or expensive investments in network hardware, such as routers and base stations, for network infrastructure. In addition, other challenges that could plague fixed infrastructure, such as network bottlenecks at key routers or base stations, do not inherently affect ad hoc networks. The cost of fixing such issues in an ad hoc network is generally less than in a traditional fixed-infrastructure network. While ad hoc networks

alleviate such problems, different types of ad hoc networks introduce additional challenges due to added needs and requirements, especially related to mobility and delay tolerance.

Mobile ad hoc networks (MANETs) are a specific type of ad hoc network in which the nodes of the network maintain varying degrees of mobility. The node mobility introduces additional challenges in regards to routing, mobility, and dealing with delay, as well as more general challenges including managing bandwidth and energy consumption for mobile devices. In particular, mobile ad hoc networks can address a range of unique situations in which ad hoc network communication is advantageous or even vital. Mobile ad hoc networks have applications as diverse as mobile social networking [11], military battlefield logistics, and sensing applications [12]. Concepts from mobile ad hoc networks are frequently used to address a similar type of networking environment – delay tolerant networks (DTNs). DTNs add another dimension to ad hoc networking – in many challenged environments, nodes that form the network may only experience intermittent and brief contact time (periods of time in which they are in wireless range to exchange data.) In this case, the chief challenge becomes designing protocols and schemes (in many cases, *store-and-forward* techniques) to route data, since there may be considerable delays until a given node can forward data. Research like [13] addresses topics for these challenged networks.

Because bandwidth and connectivity are often significant challenges in ad hoc networks, the accessibility of data can become limited if there is network congestion or high levels of mobility. One of the central technologies of this thesis, *cooperative caching*, addresses the problem of data accessibility.

### 2.1.1 Cooperative Caching in Mobile Ad Hoc Networks

As a general concept, *caching* has a broad and important role in various aspects of computing. The main goal of caching is to serve future requests for data more efficiently by storing certain data in a cache. Various caching techniques are employed through hardware and software at the kernel level, for

example, to make common operations such as disk reads more efficient. A key example of the need for caching is seen through the increased network demands on the internet, particularly through the World Wide Web. For serving HTTP and other internet-related requests faster, caching is employed to bring the relevant data closer to the user, such that the delay to deliver the response is reduced. This broad type of caching is called *web cache*. Similarly, content distribution networks (CDNs) are essentially distributed web caches that serve content across geographies, so that many geographically-diverse users can experience similar, low delays when accessing web resources.

Some caching technologies work cooperatively among caches or network devices, for example, to coordinate the caching of data of shared interest. In the Internet Cache Protocol, for example, network nodes communicate directly to discover and request data that has been cached by nearby nodes. This type of caching is known as *cooperative caching*. Most of the cooperative caching technology has been designed for fixed, wired networks. However, L. Yin and G. Cao have proposed cooperative caching schemes in [14] and [15] specifically for wireless P2P networks. Ad hoc networks can benefit from the advantages cooperative caching brings to traditional wired networks. With nodes coordinating in an ad hoc network, data accessibility can be increased by carefully caching interesting data among select nodes of the network. Figure 1 is used to illustrate a typical scenario that could occur in an ad hoc network that leverages cooperative caching:

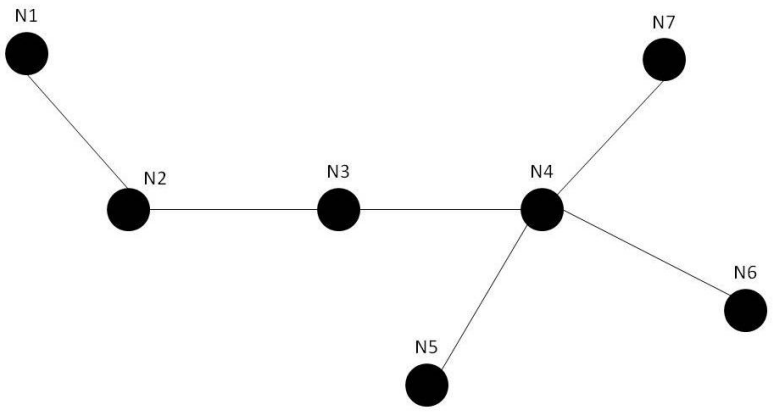


Figure 1 Example of an ad hoc network in which cooperative caching can be used.

In figure 1, node 1 (N1) is interested in a specific piece of data called D1. In an ad hoc network without cooperative caching, N1 may use a routing protocol such as AODV to request a routing path to the data server (the node that possesses and serves D1), N6. Once N1 receives the next hop information from the routing request, it may use an application-specific protocol to forward the request for D1 to N6, traversing the path N2, N3, N4, N6. At each hop, the node consults its routing table and forwards the request to the appropriate next hop. Once the request arrives at the data server, N6 responds with the data and it is forwarded back to N1. In an ad hoc network using cooperative caching, requests for data can be served by intermediate nodes along the forwarding path that have been chosen to cache the data. In figure 1, imagine that N3 has been chosen to cache D1. Instead of the request being forwarded all the way to N6, N3 receives the forwarded requests, observes that it contains the requested data in its cache, and sends the data back to N1. Because the request did not have to traverse the full path, the latency of the request was decreased – N1 was able to receive a response with less delay. In networks with mobility, there is also a greater chance that a request can be served when the number of hops to the data server is decreased.

In [14]**,** L. Yin and G. Cao proposed different schemes for coordinating cooperative cache in wireless ad hoc networks. In [15], a cooperative cache middleware was proposed to fulfill the requirements of the ad hoc cooperative cache. An algorithm for placing the caching nodes to minimize the aggregate delay in the cooperative cache was designed, and heuristics were developed to calculate this placement at the data server. In addition, a novel asymmetric approach was developed, such that processing delays on the data response path are avoided by tunneling the reply and having only the caching nodes process the data at the cache layer. Simulation results verified that this asymmetric approach outperforms other approaches due to the benefits of data pipelining.

### 2.1.2 Challenges and Opportunities

While cooperative caching has successfully been implemented on laptops, there appears to be little or no work on implementing cooperative caching on smartphones. One of the reasons for this may be due to the unique challenges that current smartphone platforms pose for cooperative caching. For one, little work has been done specifically on smartphone peer-to-peer, ad hoc networking. While there is great potential to create applications that leverage the computing power of nearby smartphones, little progress has been made in this area for namely one reason: as of May 2013, the most popular smartphone platforms (Android and iPhone, with 75.0% and 17.3% market share, respectively [16]) do not natively support creating 802.11 WiFi-based ad hoc networks. Many Android versions that manufacturers ship do not come with the capability to even join an existing ad hoc network. Many Android devices require the user to gain root privileges or manually overwrite the wpa_supplicant system file in order to be able to connect to ad hoc networks.

Therefore, for the majority of applications that would benefit from P2P networking, developers have to rely on Bluetooth or adopt a client-server architecture that seeks to mimic P2P functionality. For example, the popular smartphone application *Bump* enables users across smartphone platforms to physically "bump" their devices to trigger the exchange of user contact information and for sharing files. While it may make sense intuitively for the user devices to share data directly via a P2P communication protocol, Bump actually monitors the phone sensors to determine the physical "bump", and then it uploads the data to be shared to a cloud server. Once the server determines the two pairing devices (based upon timing and device location information, requiring user GPS data), the server sends the data to the intended client [17]. Even though Bump appears to be a P2P application, the application requires existing communication infrastructure, user location information, and a complex client-server architecture requiring internet connectivity.

Alternatively, *Bump* could have attempted to use Bluetooth to do the P2P communication without the complex infrastructure. However, a solution like Bluetooth requires a lengthy, manual device pairing

and bonding process, and low communication bandwidth (720 kbps, compared to speeds of up to 54 mbps for WiFi) [9]. And, users across platforms and devices would most likely encounter a variety of device incompatibilities that plague Bluetooth-enabled applications. Bluetooth is limited to a range generally within ten meters. For other applications requiring ad hoc networking and communication with multiple devices simultaneously, many Bluetooth devices are limited to bonding with only a single device at a time. Therefore, in order to implement ad hoc routing, additional complexity would need to be added to force devices to continuously form new pairs and to avoid dropping data when a new bonding occurs, potentially adding considerable delay to any routing protocol. It is for these reasons that Bluetooth makes a poor protocol for more demanding applications running on mobile ad hoc networks.

Due to the inherent ad hoc communication limitation on modern smartphones, there has been some work done to communicate in an ad hoc fashion without relying on ad hoc WiFi or Bluetooth, as in [18]. In this work, Trifunovic et al. use a few approaches, such as randomly scanning for open APs and other smartphones acting as mobile hotspots, to perform the opportunistic communication. However, WiFi-Opp still requires some infrastructure, such as WiFi APs. In addition, for the Android platform, tethering is not available on all versions and models. Finally, ad hoc WiFi has better throughput than WiFi-Opp. While WiFi-Opp acknowledges that Android does not universally support 802.11 ad hoc WiFi networking and that it may never, it does not provide an alternative that truly satisfies the requirement for an infrastructure-free solution. WiFi Direct, an emerging new protocol that allows devices to communicate via a pairing mechanism similar to Bluetooth, may offer some promise for P2P applications in the future. However, WiFi Direct adoption and application development has been slow, and it remains to be seen whether device manufacturers will embrace it.

## 2.2 Cooperative Cache for Smartphones

To solve the problem of data access for smartphones without 3G connectivity, a hybrid data access scheme called Cooperative Cache for Smartphones (CCS) is proposed. Assuming that an ad hoc network can be established among the nearby smartphone nodes, cooperative caching can be employed to enable data access via nearby 3G nodes, and to reduce the data access delays from far away 3G nodes by caching data throughout the ad hoc network.

### 2.2.1 Cooperative Caching Schemes

This chapter is intended to provide a review of the current cooperative caching technologies for wireless P2P networks proposed by Cao et al. in [14] and [15]. In [14], the authors present the first cooperative caching schemes for wireless networks. At that time, cooperative caching had been employed to improve the data access characteristics of the web. However, cooperative caching technologies assumed a fixed network topology, and static bandwidth on the network links – characteristics that cannot necessarily be assumed in wireless ad hoc networks. Three schemes were proposed to specifically address cooperative caching in wireless P2P networks: CacheData, CachePath, and HybridCache. These are all *passive caching* schemes. There are two types of cooperative caching: active and passive. In active caching, nodes actively communicate amongst one another to share and manage their cached data lists. In passive caching, nodes are not required to send additional control messages – instead, nodes passively inspect forwarded packets and can chose (or be instructed) to cache the data. Unlike active caching, passive caching does not add additional traffic burdens to the network. However, passive caching requires the ability to inspect passing-by packets.

In CacheData, intermediate nodes along the forwarding path cache data that has common interest among nearby nodes in the network. This way, future requests for the data can be served more efficiently

by nodes that have a shorter hop distance from the requesting node. Data is cached either when the intermediate node is interested in the data, or when the data server chooses the specific node to cache. This computation at the data server is reviewed in the cache placement section. An example of CacheData follows in figure 2 (i).

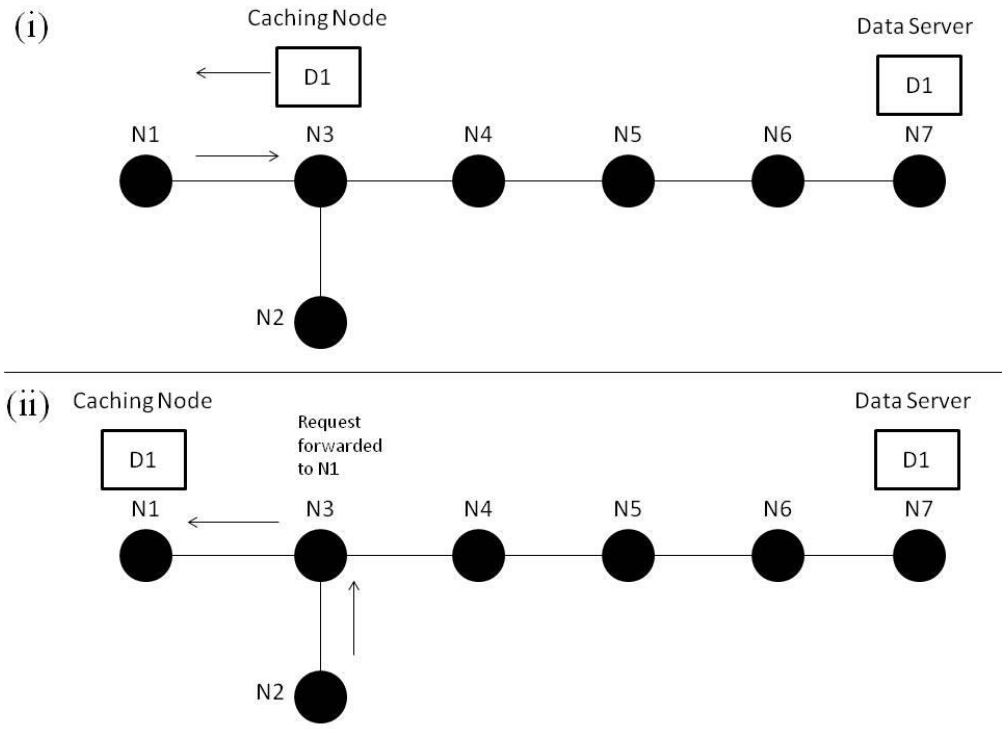


Figure 2 An example of the (i) CacheData and (ii) CachePath caching schemes.

The data request originates at N1. Assuming that the same data has been requested previously, the caching node N3 has been selected by the data server N7. Upon the data request by N1, the data has already been cached at N3, and when the request is received along the forwarding path to N7 by N3, N3 is able to check its cache and respond directly to N1. In this way, even though the original server of the data is N7, the request does not need to traverse the full path to find the node that has the data. It is this central premise that allows the data accessibility for commonly-requested data to be improved significantly.

In CachePath, path information is cached in nodes that meet a certain criteria, such that additional requests for the data can be rerouted to a node that is closer by that has the data, instead of the data server. Consider the example in figure 2 (ii). When the data request from N1 is received by N7, N7 responds by sending back the data along the forwarding path N6, N5, N4, N3, and N3 forwards the data back to N1. For example, suppose N3 decides to cache the path information. N3 records that D1 was sent back to N1. Now, suppose N2 requests D1, and sends a request on the forwarding path to N7. When the message is received by N3, N3 checks the path information that it has cached, and observes that it contains path information for D1 that points to N1. Instead of forwarding the request along the path to N7, N3 forwards the request on to N1 because it knows that N2 is closer to N1 than to N7. In general, a node caches path information when that node (in this case, N3) is closer to the caching node (N1) than it is to the data server (N7).

HybridCache is a mixed form of cooperative caching in which the methods from CacheData and CachePath are combined, such that both the data and path information can be cached simultaneously given certain criteria. This is based upon the following observations: based upon simulation and prototype results, CacheData and CachePath perform differently depending upon a variety of factors. In general, CachePath performs better when there is a smaller sized cache and when the data validity is longer (the data is not repeatedly updated, and the data remains valid). When more items are able to be cached, CacheData performs better. HybridCache takes into account these factors, and when passing-by data is processed, makes a decision to either cache the data or cache the path information. The decision process is more thoroughly described in [14].

### 2.2.2 Cache Placement

In the CacheData scheme, a node can cache data when either (i) it is personally interested in caching data and it has the available cache space, or (ii) it is chosen to be a caching node by the data server. In the case of (ii), when a data server receives a request, it computes a list of intermediate nodes along the path back to the requesting node to cache the data as it is forwarded. This computation is performed for the benefit of the entire ad hoc network participating in the cooperative cache, using global information and the path list (intermediate nodes that the request traversed). The goal of the data server is to compute an optimal cache placement, defined such that it must chose a list of nodes along the path list that produces a minimum aggregate delay. Here, the aggregate delay is defined as "the time to serve the current client request and the delay to serve future data requests coming from the same path". A greedy algorithm to solve this problem is presented in [15].

### 2.2.3 Layered Design and Asymmetric Approach

In [15], the authors propose a layered design for the cooperative cache system. Between choosing to modify the kernel to add cooperative caching functionality at the network layer and implementing the cooperative cache as its own layer (user-level process), the authors choose to implement it as its own layer. The authors note that modifying the kernel would create a number of limitations, required kernel-level memory increases due to CacheData, and the tight coupling of the cooperative cache and a routing protocol (since there is no default routing protocol used for 802.11 ad hoc mode). In designing the user space layered approach, the authors choose to implement the cooperative caching functionality such that the layer sits between the application layer and the network layer, and operates such that there is clear separation between the layered communications. One significant challenge that arises in this approach, however, is the potential to lose the benefits of data pipelining.

The current layered architecture of the internet protocol (IP) suite allows for large packets to be fragmented, such that a large packet is instead broken up into small packets. Packets can be broken up and sent, one by one, to the destination. In an ad hoc network, these smaller fragmented packets are sent from hop to hop, until they reach the destination node. At each node, the fragmented packet is examined at the network (routing) layer, and then sent to the next node. The packet does not have to be copied to the user space, and does not incur any additional delays by being processed by the upper networking layers. In this case, data transmission can benefit from data pipelining, such that the entire packet need not be assembled at each node before it is forwarded to the next node. This is because the intermediate nodes do not require the entire packet. If the cooperative caching layer of every intermediate node requires access to the packet, then any data pipelining cannot occur. This is because the cooperative caching layer, located below the application layer, would require the entire packet to be assembled from its fragments and copied to the user space for examination. This is illustrated in figure 3.

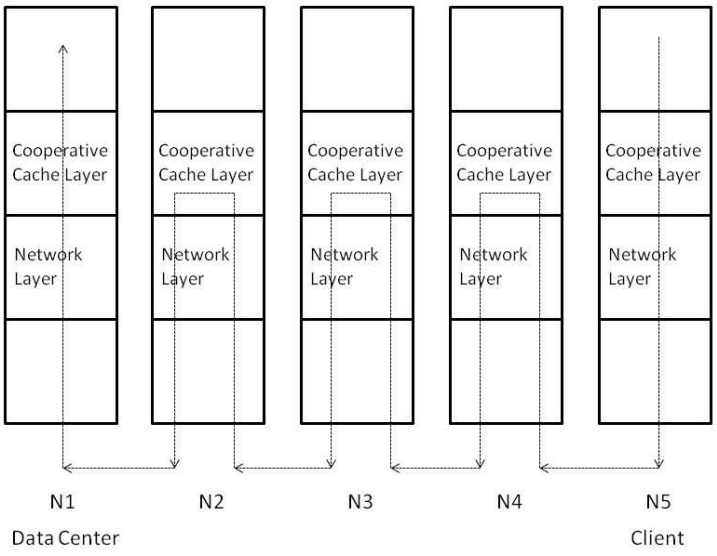


Figure 3 The layered design.

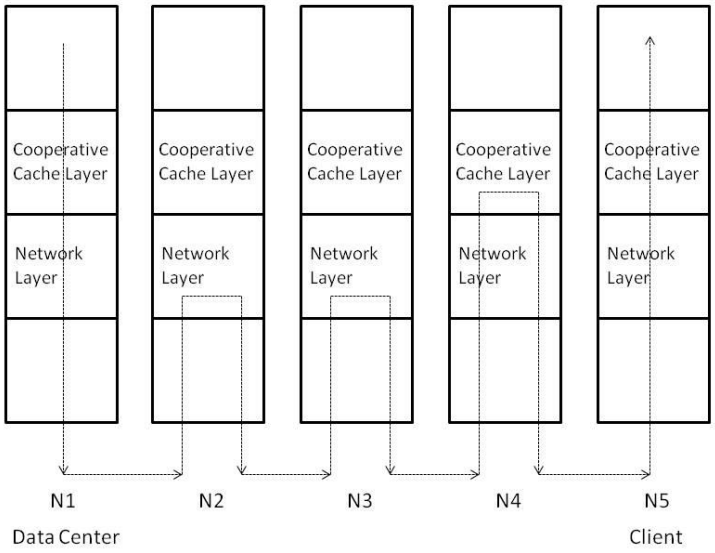


Figure 4 The asymmetric approach.

The authors propose an asymmetric cache approach to solve this problem. When a data request is sent, the cooperative caching layer needs to inspect the request in order to check if the node has a cached copy of the data that was requested. Once the request reaches the data server, a response message is sent with the data. In the proposed solution, the response packet only needs to be inspected by the cooperative caching layer at the intermediate nodes that have been chosen to cache the data. Therefore, data pipelining can still occur between nodes that are not required to cache the data. The authors refer to this as the *asymmetric approach* [15]. The data reply includes a list of the nodes to cache. This list is encapsulated, along with the data, in an application-layer packet. The data server sends the packet with the first intermediate node in this list as the destination. At this caching layer, the node removes itself from this list and sends to the next node in the list. Therefore, the intermediate nodes are not directly addressed, and only serve as intermediate routers, allowing data pipelining to occur between the caching nodes.

### 2.2.4 Current Design and Implementation

As previously stated, the cooperative caching functionality is encapsulated and implemented as a separate cooperative caching middleware layer between the application and network layers. This middleware is illustrated in figure 5.

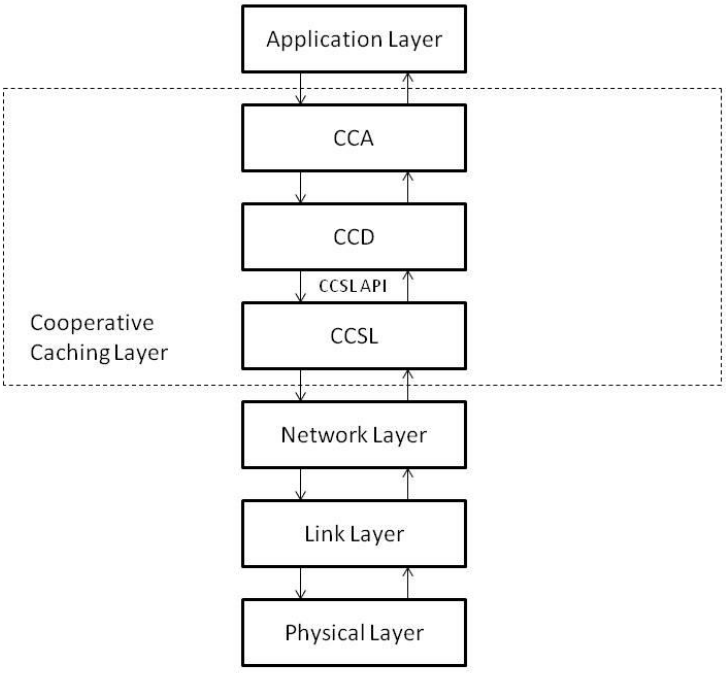


Figure 5 System architecture proposed in [15].

The *Cooperative Cache Supporting Library* (CCSL) is the component that provides interfaces to the generic cooperative caching functionality. It manages communication with the kernel routing table so that the cooperative cache schemes have access to the routing information. It is responsible for checking the passing-by packets, and manages the actual caching of data. The *cache table* stores the data access information (such as data, access frequency, etc.) and the *data cache* is responsible for storing the actual data. It is implemented as a separate user space process. The *Cooperative Cache Daemon* (CCD) is a component that implements the specific caching scheme, such as CacheData or CachePath. There is one

CCD per cooperative caching scheme, and each is implemented as a separate user space process. The *Cooperative Cache Agent* (CCA) is the component that facilitates communication between the CCSL and the user application. It is responsible for mapping application-level messages to cooperative cache messages. For example, when an application would like to request data, this is translated into a cooperative cache-level message by the CCA. Then, the CCA makes this request through socket communication to the CCSL. There is one CCA for each user application, with each one implemented as a separate user space process.

The core communication model of the cooperative caching system is the data discovery protocol. This protocol is used by the CCSL to communicate between the nodes participating in the cooperative cache. Chiefly, it allows a node to request specific data from the cooperative cache. When a node wishes to receive some specific piece of data, it generates a *data request* packet. A *data request* packet contains the following important fields:

*{source address, request id, data id, target address}*

The *source address* corresponds to the address of the node requesting the data. The *request id* is a sequence number for the data request. The *data id* is a unique identifier for the specific data. In the implementation, each data is keyed by a unique string. The *target address* is the address of the node known to be the data server. After the *data request* packet is generated, it is sent to the next hop on the path to the data server. That node's CCSL consults its cache to check if it contains the data. This process continues until the data is found at an intermediate node, or the *data request packet* reaches the data server. Once a node is reached that has the data specified by *data id*, the node responds with a *data response* packet. The *data response* packet contains the following important fields:

{*source address, request id, data id, target address, data size, lifetime, data version, data*}

The *source address* is the address of the node that has requested data. The *request id* corresponds to the *request id* contained in the respective request packet. The *data id* is the unique id of the data. The *target address* is the address of the data server. The *data size* is the size of the response data in bytes. The *lifetime* refers to the length of validity of the data, giving the receiver a sense of how long the data will remain valid. The *data version* is a record of the current version of this data. The *data* field itself is the actual response data. In addition, the *data response* packet also contains an optional *cache_list* field, which is a list of the caching nodes that were selected by the data server. Upon arrival of the *data response* packet at each of the intermediate caching nodes, the caching node removes its address from the *cache_list* field and forwards the packet to the next node address in the list.

If no data called *data id* can be found in the cooperative cache, or there is some other processing error at an intermediate node or the data server, a *data error* packet is generated and transmitted back to the requesting node. This packet contains fields identifying the request (ie. *request id, data id)* and information regarding the nature of the error, such as error type, description, and the node reporting the error.

## 2.3 System Architecture

### 2.3.1 Architecture

The system architecture of the proposed cooperative cache layer for smartphones remains similar to the proposed architecture in [15]. We continue to use the layered approach, such that the cooperative cache behaves as its own self-contained layer between the application and network layers. The significant components of the cooperative cache layer include the Application Agent (AA), the Cooperative Cache Service (CS), the Interface Selector (IS), and the AODV Routing Module (RM). There are some

architectural changes proposed due to the difference in target platform which will be introduced here, and expanded upon in Section 2.4.

Chiefly, the library that implements the cooperative cache schemes, logic, caching, and so on has been consolidated into a single component called the Cooperative Cache Service. The interface selector has been introduced to allow the service to switch between 3G and WiFi, depending upon the available network. To realize this implementation, the platform must offer interfaces to the communication hardware – this has been represented as the Android Application Framework (but could also be the platform/framework for another smartphone platform such as iOS.) Finally, due to the challenges of performing ad hoc routing on the Android platform, we include a component that implements AODV ad hoc routing. The advantages of this approach will be addressed in Section 2.3.5. While the basic architecture remains very similar to the system architecture of [15], the implementation details and some design challenges differ due to the Android smartphone platform and hybrid data access.

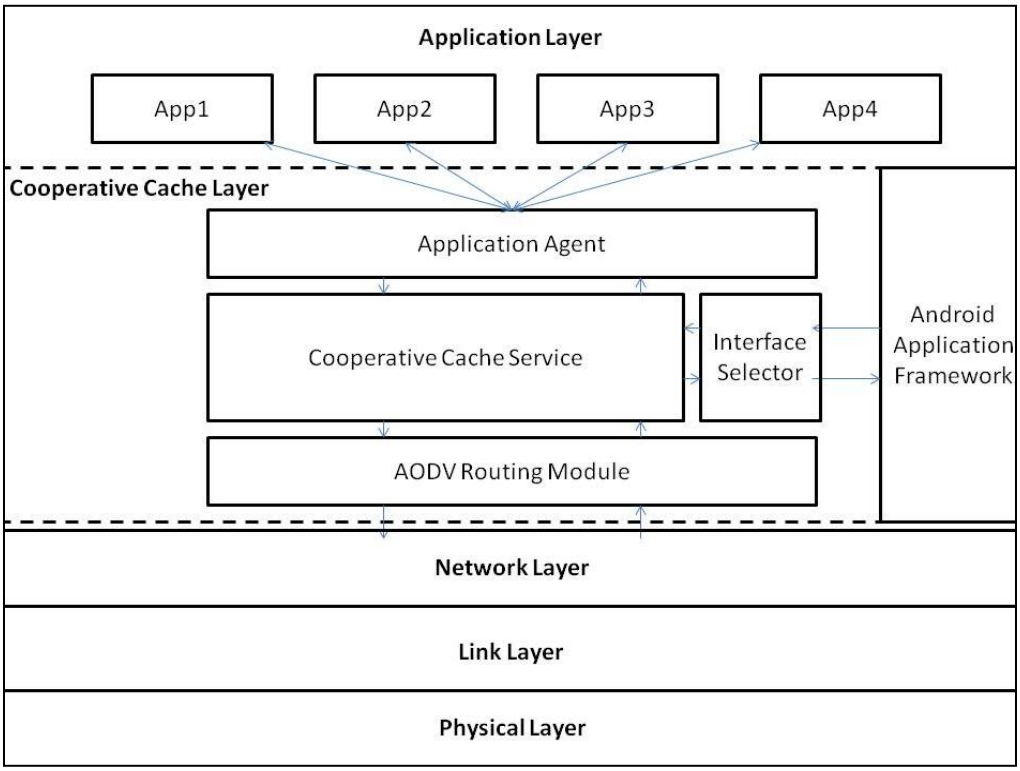


Figure 6 Cooperative cache for smartphones system architecture.

### 2.3.2 Application Agent

The application agent's role is to facilitate communication between the applications and the cooperative caching service. The AA meets three basic requirements for properly facilitating this communication: first, it offers an interface to the application (through the Cooperative Cache Helper Library, discussed in Section 2.4.4) to request data from the CS. Second, it stores and manages the applications using the cooperative caching by recording application package names, socket information (if still valid), and active status. Using this information, it can route the data response back to the appropriate application. Finally, it maintains an active connection with the CS to request and receive data.

### 2.3.3 Cooperative Cache Service

The Cooperative Cache Service is responsible for implementing the logic pertaining to the cooperative caching, as well as the storage and management of the cache itself. This includes implementing the CacheData, CachePath, and HybridCache schemes. The CS offers an interface to the application agent to request data from the cooperative cache. With the exception of hybrid data access via the interface selector, the CS is functionally-equivalent to the combined roles of the Cooperative Cache Daemon and the Cooperative Cache Support Library in [15]. Similarly, the CS maintains the *cache table* and *data cache*, checks passing-by packets, and interfaces with the routing module to obtain routing information. The interface that it provides to the applications is via the AA. The CS allows the AA to request data via a simple *request_data* function, and routes data to the application by calling a *receive_data* function that the user application can listen on.

### 2.3.4 Interface Selector

The Interface Selector's purpose is to select an interface to request data on demand. If 3G is available, the interface selector will select 3G and the request will be performed over 3G. However, in the case when the user does not have a 3G plan or otherwise cannot access 3G, the interface selector is responsible for switching interfaces to ad hoc WiFi.

### 2.3.5 AODV Routing Module

Due to implementation constraints, a necessary tradeoff to support ad hoc networking is to include the routing at the application layer. This way, the kernel does not need to be modified to support ad hoc routing. In addition, the ad hoc network can function as an overlay – only including nearby nodes that are participating in the ad hoc network for the specific application, for example. Finally, the routing module can be customized and extended to specifically support the needs of the cooperative cache, such as adding additional protocol messages if desired.

## 2.4 Design and Implementation

### 2.4.1 Previous Design Issues

In [15], a number of design issues are presented when considering the design of a dedicated cooperative caching layer. The design decisions made there are consistent with the considerations made for the smartphone implementation. In this implementation, the layer is again implemented in the user space for similar reasons – it allows greater flexibility, and does not violate the layered approach. The

data discovery protocol is consistent with the data discovery protocol in [15], such that smartphones and other devices can participate in the same cooperative cache. Finally, the asymmetric approach is again used to reduce the time spent in the intermediate nodes copying messages between the kernel and user space.

### 2.4.2 Implementation

The smartphone cooperative cache layer is implemented specifically for Android, although there may be some similar abstractions for operating system constructs that can be leveraged for other platforms.

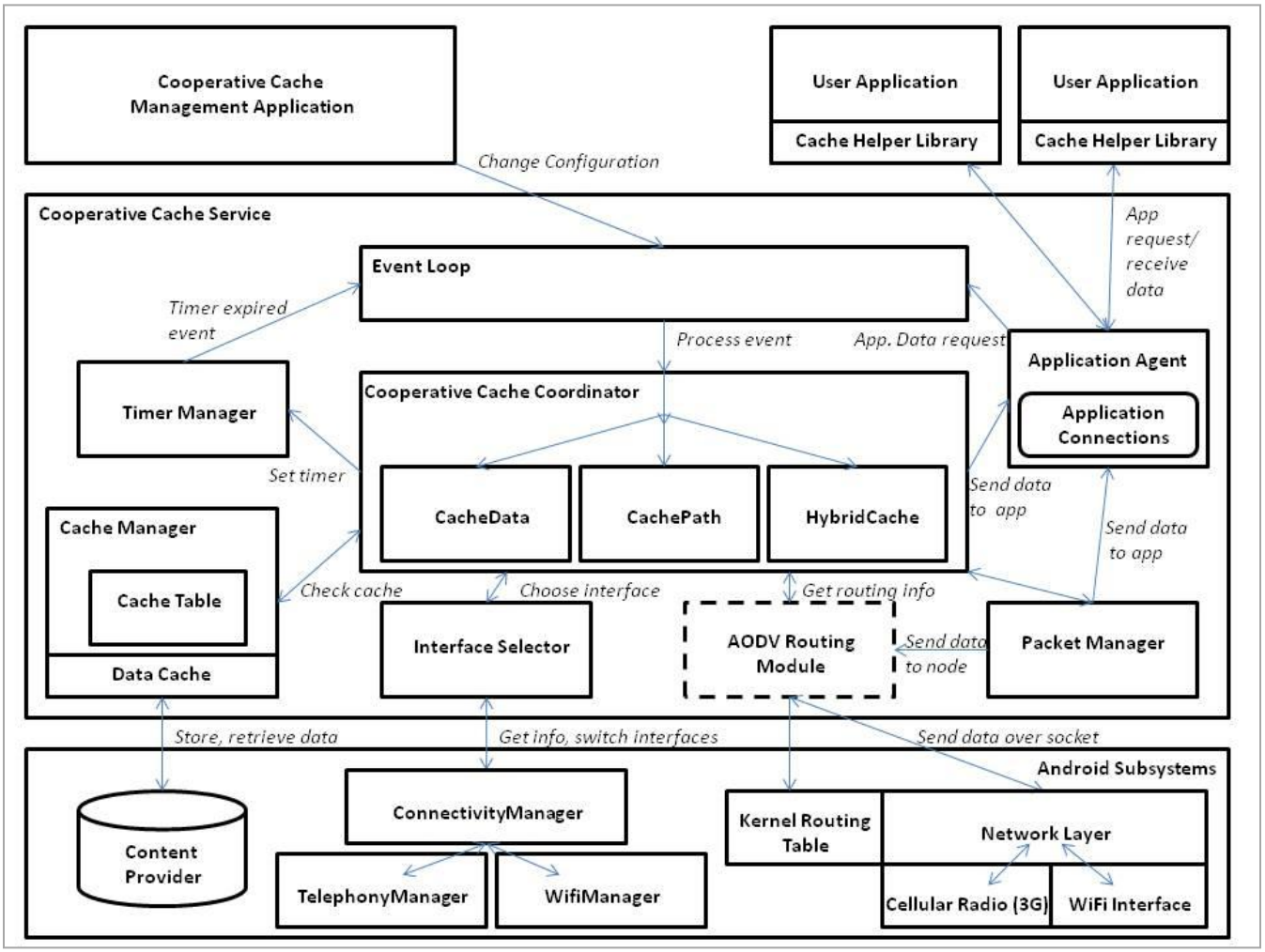


Figure 7 Component diagram of the cooperative cache service for smartphones.

### 2.4.3 Cooperative Cache Service

The cooperative cache layer is implemented as a persistent Android Service. A Service [19] is an abstraction for a long-running task or series of tasks. In this case, the Service utilizes a single thread in a separate process to perform the chief cooperative cache functionality, communicate with the user applications, and use the communication interfaces. While a Service shares similarities with a thread or process, the lifetime is largely controlled by the operating system. If the operating system is running low on memory, it can kill the service – therefore, data stored in memory is periodically serialized to disk to ensure that no data is lost. The Service architecture and related components are illustrated in figure 7. Cached data is stored in a Content Provider [20], which is an Android abstraction for a file-backed database. The AODV Routing module implements the AODV routing at the application layer, since AODV is not natively implemented for Android. This component has been adapted from [21].

### 2.4.4 Cooperative Cache Helper Library

The library that is distributed to build applications leveraging the cooperative cache layer is called the Cooperative Cache Helper Library. The helper library can be instantiated and embedded into the user application, and provides a simple interface for requesting and receiving data. It runs within its own thread in the user application. For the lifetime of the user application, the helper library maintains an open socket connection with the Cooperative Cache Service's Application Agent. The chief function exposed to the application developer is *request_data(id)*, which allows the user application to asynchronously request data from the cooperative cache. Extra parameters are available, allowing the user application to specify the communication interface, destination device, a timeout period, a server URL (for 3G access), and so on. In addition, the user application can register a callback function for two events: *on_data_received(id, data)* and *on_error(id, error).* These functions are called asynchronously in a

separate thread when either data is delivered back to the application, or there was an error in retrieving the requested data.

### 2.4.5 Application Agent

The Application Agent manages the communication between the applications (more specifically, the instantiated helper libraries) and the Cooperative Cache Service. It facilitates this communication by managing a list of socket connections between the applications and the Service, keyed by the unique Android application package name such as "cse.psu.freilich.myapplication". When a user application finishes and the socket connection is closed, it can still deliver data to the application. If the socket connection has been closed, it constructs an Android Intent [22] with the URI [23] pointing to the Content Provider location with the received data. When this Intent is fired, the operating system can trigger the user application to start and retrieve the data.

### 2.4.6 Interface Selector

The Interface Selector component is responsible for choosing the interface, as well as managing the active communication interface. A BroadcastReceiver [24] is registered to be able to learn about connectivity changes and act upon these changes. The two important actions that are registered are NETWORK_STATE_CHANGED_ACTION and CONNECTIVITY_ACTION. NETWORK_STATE_CHANGED_ACTION notifies the receiver that the state of WiFi connectivity has changed, and we can check if a new network has been connected to. The CONNECTIVITY_ACTION is a generic action indicating that some connectivity state has changed. Here, we can check if the network type is of TYPE_MOBILE and that the network is connected. Upon either connecting to 3G or WiFi, an

event is generated and sent to be received by the event loop. Then, the appropriate action is taken, such as using 3G to request the data, for example.

## 2.5 Evaluation

A test bed has been constructed to evaluate the CCS scheme in comparison with the regular ad hoc data access. The test bed is composed of five Samsung Nexus S (GSM version) smartphones running Android 4.0.4. Each device has a 1 GHz single-core processor, with 512 MB RAM and 16 GB internal storage. The batteries are 1,500 mAh rechargeable Lithium-ion batteries, consistent with other contemporary smartphone batteries. The five smartphones have been arranged in a fixed ad hoc network with the network topology shown in figure 8.

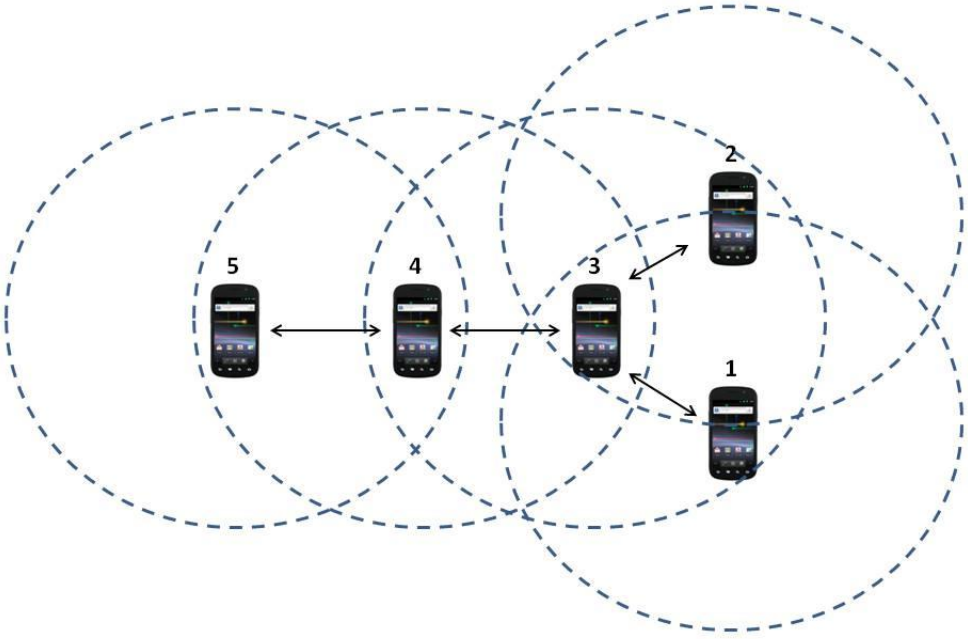


Figure 8 Test bed topology.

The test bed is performed by initiating a fixed schedule of requests at node N1, and the data center with 3G access is node N5. In addition to this fixed schedule, requests are randomly generated at N2 to simulate additional cooperative caching activity in the network. Before the test bed is run, 100 unique pieces of data have been generated with random sizes ranging from 1 KB to 50 KB. 80 requests are assigned to N5, and the remaining 20 requests are randomly assigned to the other nodes. Next, 5 procedures have been randomly generated as follows: for each procedure, we generate 50 random data requests that are to be performed in sequence. Between each request, there is a short, random waiting period of between 1 to 5 seconds. These 5 procedures (250 requests in total) are run for each of the schemes that are evaluated. Because smartphones are likely to be mobile, we focus the evaluation on the CacheData scheme. In the following tests, the regular scheme refers to retrieving the data from N5 through the ad hoc network without caching. Measurements are gathered on the requests initiated at N1.

In the given topology, data can be retrieved from up to 3 hops away. For N1, retrieving data directly from N5 requires the request and response to travel 3 hops. If data is retrieved from the cache of a closer node like N3 or N4, the number of hops is reduced. Intuitively, each additional hop to retrieve the data incurs an additional access delay penalty. While data can be retrieved at a shorter distance when using the cooperative cache, without the cooperative cache the data can only be retrieved from a distance of 3 hops (N1 to N5). Although, the delay for N5 without using cooperative caching is slightly smaller than with using cooperative caching, since there is less cache layer processing overhead.

Figure 9 shows the average data access delay of the regular scheme compared to using cooperative caching. In this case, each cooperative cache node is given enough cache space to store all of the data without evicting any cache entries.

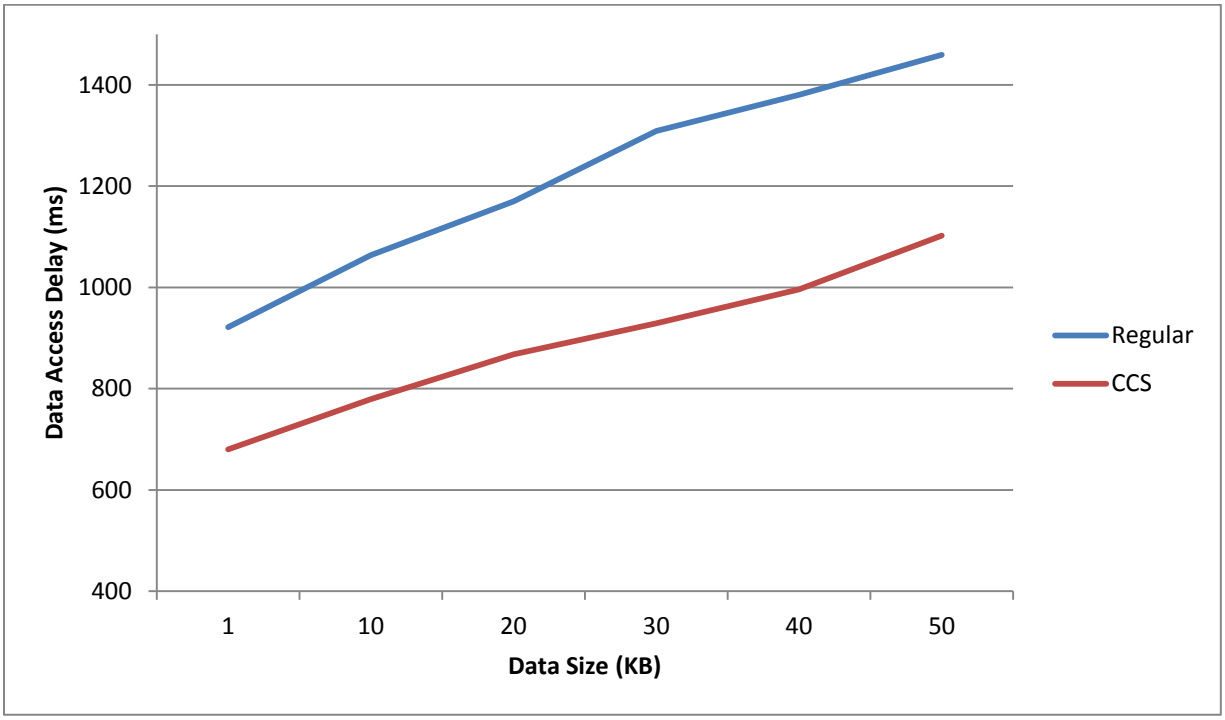


Figure 9 Access delay as the data size varies.

Because each node has adequate space, the cache hit rate in these tests is between 40% and 50%. Regardless of the data size, the CCS method consistently saves between 25% and 30% of the delay over the regular method for the given topology and workload. The measurement of delay for the regular approach also assumes that the 3G node (N5) possesses the data before it receives the request from N1. In practice, if N5 does not possess the data, then significant additional delay could be added since N5 would have to request the data over 3G before it can send it back to N1. Because the cooperative cache technique reduces the number of requests that must arrive at N5, cooperative caching would reduce delay even further in practice than has been measured here.

While both the regular and CCS method increase delay when the data size increases, the CCS increases delay about 22% less than the regular scheme over the interval from 1 KB to 50 KB data size. This is because, as demonstrated in table 1, the effect of increasing the data size is amplified as the number of hops to the data is increased. When there are additional hops to retrieve the data, it must incur additional processing and transmission delays according to the number of hops. Conversely, when the

cooperative caching is used with adequate cache space, increasing the data size does not increase the number of hops required to retrieve the data. Therefore, cooperative caching benefits even more when data can be retrieved with fewer hops, even when the data size increases.

Table 1 Cooperative cache access delay when hop count and data size vary.

| Cooperative Cache Access Delay (ms) | | | | | | |
|---|---|---|---|---|---|---|
| | 1 KB | 10 KB | 20 KB | 30 KB | 40 KB | 50 KB |
| 1 hop | 410.6 | 416.2 | 428.8 | 476 | 521.4 | 595.33 |
| 2 hops | 782.3 | 806.4 | 954 | 980.2 | 1005.8 | 1189.4 |
| 3 hops | 990.7 | 1112.2 | 1237.1 | 1342.3 | 1414.7 | 1568 |

In the test cases where adequate caching space was allowed, about 30% of data was retrieved within one hop, 50% within two hops, and the remaining data was retrieved from the data server N5. When the cache size is restricted, data must be evicted and the cache hit rate is reduced. Therefore, the access delay depends strongly upon the ability of the nodes to cache the data. In figure 10, the access delay is shown for varying cache sizes when data items of size 50 KB are requested and cached throughout the network.

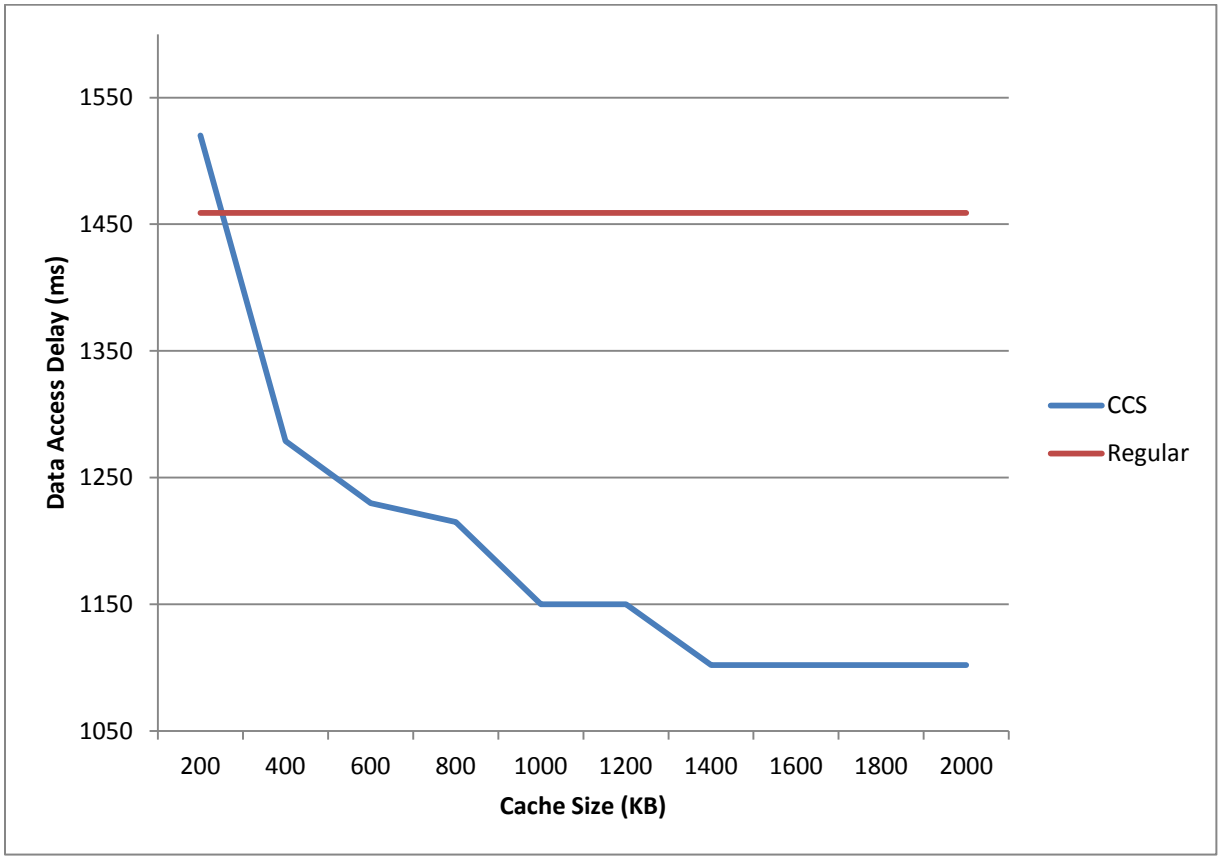


Figure 10 Access delay as the cache size varies.

As the cache size increases, the hit rate increases (from a few percent to upwards of 50% as all of the data of common interest can be cached.) When the cache size is the smallest, only 200 KB or 4 data items can be cached at each node. As a result, the cache hit rate is very small and the cooperative cache overhead outweighs any of the small benefits from caching. When the cache is of adequate size when it is larger than 1.2 MB, all of the needed data can be cached without evicting any of the data that N1 was interested in. In that case, accessing data from the cooperative cache is much likelier to produce a cache hit and a smaller access delay. In this case of relatively small data items and number of requests in the network, the cache size can be relatively small and still be very effective. An effective cache size depends upon the application requirements and activity within the network. When the cache data is persisted to disk, then the cache size can be adjusted to fit applications with large data requirements, since many

modern smartphones contain relatively large amounts of storage. Common applications on Android routinely cache megabytes of data, and contain even more data stored in databases.

# Chapter 3

## Request Aggregation-Based Scheme

While we can address situations in which smartphone users do not have mobile data plans or are otherwise unable to use 3G data, there are also times when many smartphone users have access to 3G data and are in close proximity. Consider an example of a group of students on a long, five hour bus ride for a field trip. Each student has his or her own 3G data plan and a 3G enabled smartphone. On this bus ride, each of the students wishes to use his or her smartphone to work on homework and browse the web. Unfortunately, because there are no power sources on the bus, the students must consider how 3G usage will affect the battery lives of their smartphones. The typical web browsing habits, for example, can be very energy and delay inefficient. If a user requests data somewhat frequently (for example, every 30 seconds), then every request can suffer from high tail energy consumption and repeated 3G network connection delays of up to 2 seconds.

To avoid these energy and delay problems, the students decide that they can work together. They choose to elect a student (perhaps the one with the smartphone that has the most battery) to act as the "leader". The leader communicates with the other students, writes down the data that the students want, performs all of the requests on his or her smartphone, and then transmits the data back to the student that originally requested it - over Bluetooth, for example. This way, the other students can avoid having their devices consume energy using their energy-heavy 3G interfaces, and the leader can perform all of the requests and then quickly get back to work.

However, there are some severe limitations with this otherwise intelligent approach. First, how many students could this strategy support without overwhelming the leader? With more students, the coordination becomes more complicated, and it will take longer for each student to receive his or her requested data. In addition, how can the leader manage the status of all of the requests – if a student asks

how long his or her request will take, how can the leader know this? How can the student make a judgment about whether or not it would be helpful to have the leader perform the request instead of the student performing it?

To address the energy and delay problems discussed above, a method is proposed for aggregating smartphone data requests to reduce the data access delay and the energy consumption of the requesting devices. In this request aggregation scheme, a cluster of smartphones leverages a clusterhead to perform the requests on behalf of the smartphones. The smartphone nodes in this one-hop ad hoc network ping the clusterhead for its interface state and request queue time; if energy and delay can be saved, then the requests are forwarded to the clusterhead to be fulfilled over 3G. This approach has two distinct advantages; first, the network nodes can benefit from better data access delays by leveraging the 3G interface state of the clusterhead and can avoid wasting energy on their own 3G interface usage. Second, because the clusterhead is performing the requests for the other nodes, it experiences less delay and less wasted tail energy for its own requests since it is more likely to already be in the high energy state. Using this approach, both the network nodes and the clusterhead benefit from the aggregation of requests.

## 3.1 Background and Related Work

Effective management of the 3G radio is one of the leading challenges for improving smartphone battery life and reducing internet access delays. When used incorrectly, the 3G interface can be a significant source of unnecessary energy consumption and delays. The power and delay characteristics of 3G depend on two main aspects: the 3G Radio Resource Control protocol and the transmission delay and energy. The 3G Radio Resource Control protocol defines how the device and cellular backbone network can efficiently use the network resources. As a result, three energy states are defined for the device – typical energy state times and values are displayed in figure 11.

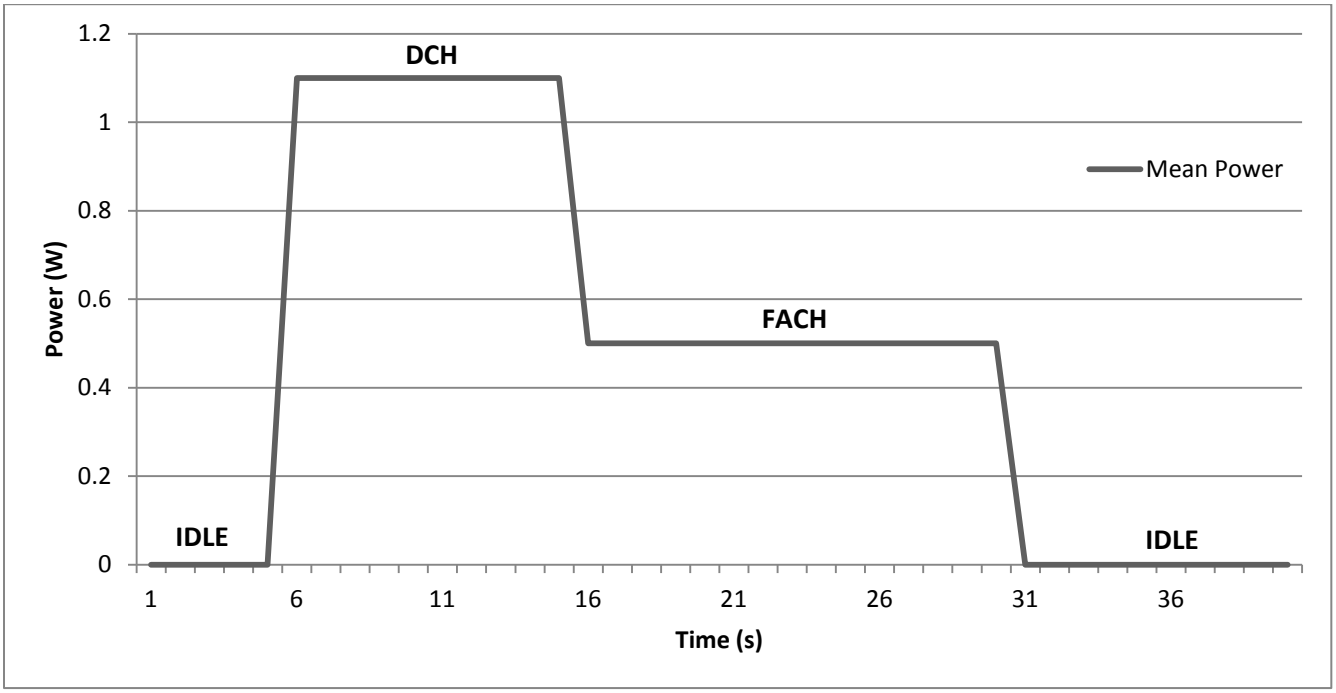


Figure 11 Typical mean power usage for the 3G interface.

In the IDLE state, the device does not have a connection to the backbone network. From the backbone network's point of view, no significant resources are being consumed by this device. However, a device cannot perform any signaling or data transfer when in the IDLE state. In the FACH state, the device can access the network via low-speed, common shared transmission channels. In the DCH state, a dedicated channel is utilized by the device to transfer data. The DCH state is required to create the connection necessary for user data transmission. In addition to these requirements, the different channels also consume energy at different levels. The IDLE state is a low-power state. The DCH state requires significant power to hold the dedicated channel, and uses the most power of the states. Finally, the FACH state is a mid-level power state that requires about half the energy used in DCH. The interface automatically transitions between the three states based upon the level of transmission activity.

To control how the interface state transitions, inactivity timers T1 and T2 are defined. T1 is the timer that moves the interface state from DCH to FACH. This timer is reset to 0 whenever a data transmission begins, and starts when a data transmission ends. If the timer reaches its threshold value, it switches the state from DCH to FACH to signify that transmission of data has stopped and that the dedicated channel may no longer be needed. When the FACH state is entered, T2 begins. When T2 expires, the state is moved from FACH to IDLE. This period of inactivity is known as the tail time. While different networks may have different values for the T1 and T2 inactivity timers, T1 and T2 are commonly triggered at 4 and 15 seconds, respectively.

The behavior of these state transitions can become a significant source of delay. This is primarily due to the delay incurred from connection establishment. When moving from IDLE to DCH, data cannot be transmitted immediately. First, the connection must be made with the backbone network and the various resources acquired in order to begin transmission. This process usually takes between 1 and 2 seconds. Similarly, to move from the FACH to DCH state, there is about a 1.5 second connection delay. However, when the device is already in the DCH state there is no connection delay since there is already a connection established with the backbone network. Therefore, in the DCH state data can be transmitted immediately.

The state transitions can also incur significant, commonly unnecessary energy consumption. One especially important issue is known as the tail energy problem. The energy consumed over the duration of T2 is referred to as the tail energy, since it is the energy used at the end of the request. On the one hand, this state is useful for the backbone network, as it keeps the device in a medium power state that can perform the network connection faster. From the device perspective, however, the tail energy is often wasted energy. Once the transmission is complete and T1 is triggered, another data transmission may not take place for a significant amount of time. For a value of T2 such as 15 seconds, this means that the device may spend 15 seconds in a medium power state with no added benefit. For typical data transfers,

the tail energy can consume upwards of 60% of the total energy used for a 3G transmission, often with no benefit to the user.

For many users, their behaviors cause the 3G interface to behave inefficiently. For example, if a user requests data over 3G every 30 seconds, then each request can require a full promotion from IDLE to DCH, which causes the 2 second connection delay and will most likely incur the full tail energy. If the user requests 5 pieces of data – 1 piece every 30 seconds - there would be 5 connection delays and 5 full tail energy consumptions incurred. If the user were able to aggregate these requests and perform them all at once, then only a single connection delay and one energy tail would occur.

Much research has been done to improve the energy and delay performance of the 3G interface and backbone network. In [25], the TOP protocol is proposed to have the handset notify the network to release resources on an idle tail. In this way, TOP modifies the inactivity timers to reduce the tail time. It uses a related modification of 3G called fast dormancy [26] to reduce the tail time. However, this approach requires a network that supports fast dormancy and the ability to modify the inactivity timers. In [27], TailEnder is proposed to reduce the 3G energy consumption. It requires some delay tolerance to aggregate and schedule requests so as to use the 3G interface more efficiently.

### 3.2 Request Aggregation-Based Scheme

To alleviate the high delays and largely unnecessary high-energy tails that occur during the normal operation of 3G, the Smartphone Request Aggregation Scheme (SRAS) is proposed. When smartphones are arranged in a cluster (every node is within 1 hop of another), we can leverage an ad hoc network of smartphones to offload requests to a clusterhead. To start, a node is chosen by some process (for example, voting or random selection) to become the clusterhead (CH). As the CH, the node's responsibility is to perform requests that the other nodes send to it over ad hoc WiFi. Therefore, the CH

keeps its WiFi interface on and is ready to receive data. The other nodes in the cluster are called cluster nodes (CN). Once a CH has retrieved the requested data over 3G, it can forward the data back to the requesting CN.

### 3.2.1 Decision Phase

Before a node is ready to request data from the CH, it needs to determine if it is favorable to do so. The first part of the SRAS scheme is when the CN sends the CH a message called a *ping request*. The purpose of the message is to inform the CH that the particular CN is interested in offloading a data request, and would like to ping the CH so it can determine if it should send the CH the data request. In response, the CH sends a *ping reply* with the current 3G interface state, the values of T1 and T2, the expected request waiting time, and the time that the *ping reply* was transmitted.

When the CN receives a *ping reply* packet from the CH, it can choose either to offload its request to the CH, or use its own 3G interface to perform the request itself. There are two aspects to consider: whether offloading the request will save access delay, and whether offloading will save energy consumption. Besides abnormal cases of extremely large or long requests (such that the CN has to maintain its WiFi interface for long periods of time while waiting for a reply), it is usually more energy efficient to offload the data request.

#### *Cluster node local request*

The cluster node local delay is the delay the CN would incur to request the data over its own 3G interface.

$$Delay = connection\ delay + data\ transfer\ delay$$

The connection delay is 1.5 seconds if the current state is FACH, and 2 seconds if IDLE. The data transfer delay can be calculated given the 3G rate and the size of the requested data, by

$$Data\ transfer\ delay = data\ size/3G\ rate$$

Additionally, network conditions such as round-trip time can also be factored into the data transfer delay. To calculate the energy required to perform the 3G transfer locally, the model from [27] can be used. Specifically, the 3G transfer energy for a transfer of $x$ bytes can be calculated by

$$R(x) = 0.025(x) + 3.5$$

where $R(x)$ includes the energy required to bring the interface up to the high energy state, the transfer energy, and the tail energy.

***Forwarding to the clusterhead***

To forward the request to the CH, the total delay incurred includes the delay for the *ping* messaging, the delay for the CH to perform the request over 3G, and the time to transmit the data back over the WiFi interface to the CN. Processing delay for the CH is also considered. In addition, because the CH must perform 3G requests one-by-one, there may be queued requests waiting to be performed, creating a waiting period before the given request can be completed.

$$Total\ delay = ping\ delay + waiting\ time + connection\ delay + data\ transfer\ delay + reply\ delay$$

The ping delay is the delay to transmit, process, and receive the ping response.

$$Ping\ delay = ping\ request\ size/WiFi\ rate + ping\ processing\ delay + ping\ response\ size/WiFi\ rate$$

The *waiting time* is the time that it will take to fulfill the queued requests before the current request can be performed. This is the summation of the data transfer delays for the queued requests. The *connection delay* depends upon the state of the CH's 3G interface. If it is IDLE, then the delay is 2 seconds. If it is FACH, then the delay is 1.5 seconds. If the state is DCH, the connection delay is 0 seconds. Or, it could be the time remaining if the connection has already initiated.

$$Data\ transfer = data\ size/3G\ rate$$

The reply delay is the delay incurred from transmitting the data request packet, the request processing delay at the CH, and the delay to transmit the data back to the requesting CN.

$$Reply\ delay = data\ request\ size/WiFi\ rate + request\ processing + data\ reply\ size/WiFi\ rate$$

To calculate the energy required for the CN to offload the request, the WiFi data transfer and maintenance energy must be calculated. The model from [27] can be used to calculate these. For the transfer energy,

$$R(x) = 0.007(x) + 5.9$$

where *R(x)* is the energy due to data transfer and scanning/associating for WiFi. To be precise, the energy component for scanning and associating can be removed from this calculation. In addition, the maintenance energy was measured to be 0.05 J/sec. To calculate the maintenance energy:

$$WiFi\ Maintenance\ Energy = WiFi\ On\ Duration \times 0.05\ J/sec$$

where the WiFi on duration is the time that the WiFi interface is left on. Specifically, this is the time between sending the *ping request* and receiving the *ping reply*, and the time between sending the *data request* and receiving the *data reply*. This can be calculated using results from the CH request delay. To calculate the entire WiFi energy,

$$WiFi\ energy = ping\ transfer\ energy + data\ transfer\ energy + maintenance\ energy$$

The CN makes the decision whether or not to offload the request by comparing the expected delay to offload with the computed delay to perform the request locally. If the delay to forward the request to the CH is less than the local delay, it constructs a packet to forward this request to the CH. In addition, if the CN only wishes to save power and can tolerate some additional delay, the CN can choose the most energy efficient method.

### 3.2.2 Request Phase

If the CN decides to offload the request to the CH, the CN sends a *data request* to the CH indicating the particular CN wishes for the CH to request the data on its behalf. On receipt, the CH saves the CN's address so that it can return the response back to it. Next, if the CH is not in the DCH state, it moves to this state and begins to request the data over 3G. If there are queued requests, the request is

added to a queue of waiting requests. Once it reaches the head of the queue, the request for the given URL is performed over 3G. When the CH has the data, it constructs a *data reply* packet and transmits the data back to the CN over ad hoc WiFi.

## 3.3 Evaluation

To assess the performance of the proposed scheme, SRAS has been implemented and evaluated via simulation. For the tests, requests are generated randomly every 15 to 25 seconds at each node. Each test is a simulated run of the system for 10 minutes. An ad hoc WiFi rate of 54 Mbps is assumed, and the 3G rate is varied between 1 and 3 Mbps. The processing delay at the CH is 50 milliseconds for a ping request, and 100 milliseconds for a data request. The network round trip time for requests over 3G is 100 milliseconds. For the 3G Radio Resource Control protocol, it is assumed that there is a 2 second delay to connect when moving from IDLE to DCH, and a 1.5 second delay when moving from FACH to DCH. T1 triggers at 4 seconds, and T2 triggers at 15 seconds. SRAS is compared against the regular data access, where each node independently switches to 3G to request its own data.

### 3.3.1 Cluster Nodes

As the number of CNs increases, the delay to access data from the CH decreases, as long as the CH's request queue remains short. When there are more CNs, there are a greater number of requests arriving at the CH. When the number of requests reaches the frequency such that the CH's 3G interface remains in the DCH state, every request is able to be retrieved very quickly – as the CH's interface remains connected to the cellular network and does not require any connection delay. Therefore, the only delay for a request depends upon the data size, data rate, and the network round trip times. In the following analysis, the performance of SRAS is compared with the performance of the CNs when they do

not use a CH to offload requests. Figure 12 shows the average access delay for requests as the number of CNs increase.

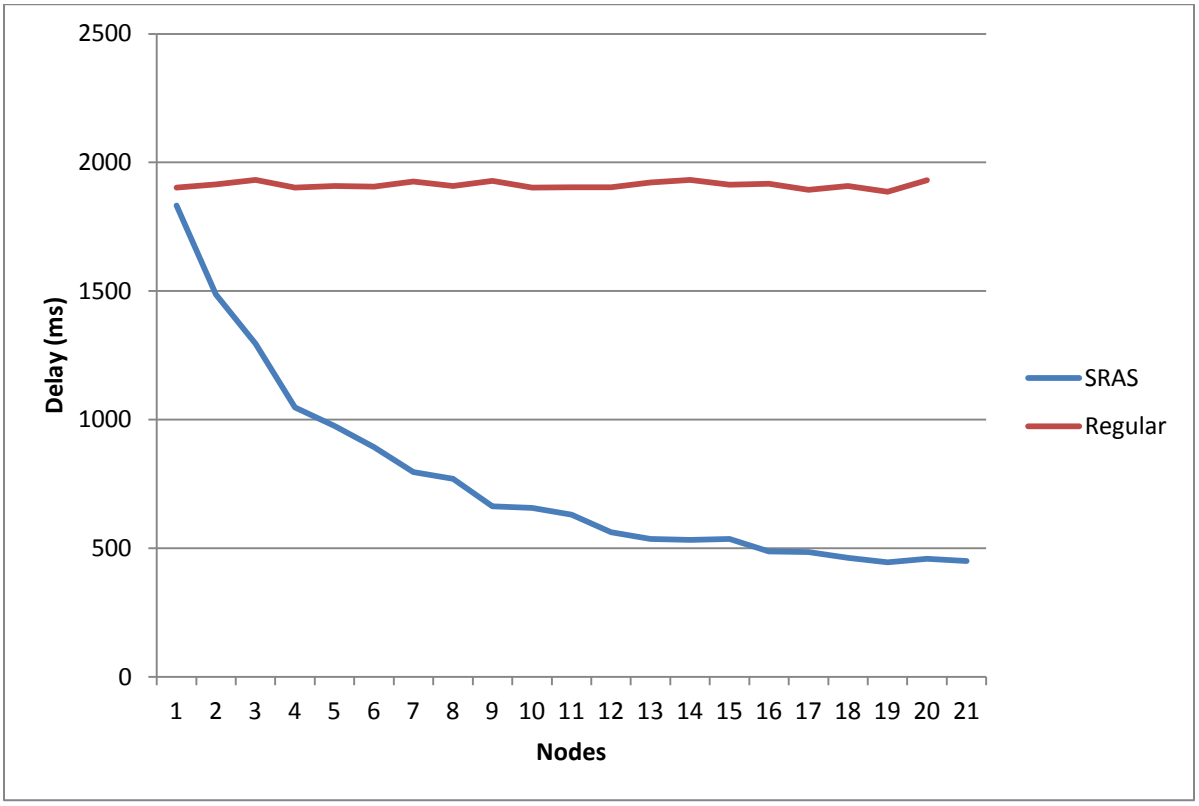


Figure 12 Average delay per request at the cluster nodes.

In figure 12, the average request delay is measured as additional CNs are added to the cluster, where the 3G rate is 3 Mbps and the data being requested is 50 KB. Between 1 and 10 cluster nodes, the access delay varies significantly. When there are few nodes, the CH frequently moves from the DCH to the FACH state, since the T1 timer threshold is only 4 seconds. Once this move occurs, there is a delay penalty for the CH – it takes 1.5 seconds to switch from FACH to DCH. However, even when there are few nodes and the 1.5 second FACH to DCH delay is incurred, it is still advantageous for the CNs to offload requests to the CH. This is because, unless the CNs are requesting over 3G fast enough, they will experience a similar delay to switch from IDLE or FACH to DCH. Once a CN starts offloading requests

to the CH, it will likely experience the full 2 second connection delay if it decides not to offload because its 3G interface is in the IDLE state. As long as queue times do not overly burden the CH and the CH's 3G state remains in DCH, it is always favorable for the CNs to offload their requests to the CH.

It is important to note that for the CNs, SRAS will always perform better than not using SRAS. This assumes that the CH can accurately calculate its queue time. Even if the queue time of the CH is large, a CN can perform the request on its own and still do just as well if SRAS is not used, which is the purpose of using the ping request/reply to inform the CN of the CH status and queue time. Still, if it offloads *any* of its requests to the CH, it will perform better than if SRAS were not used at all.

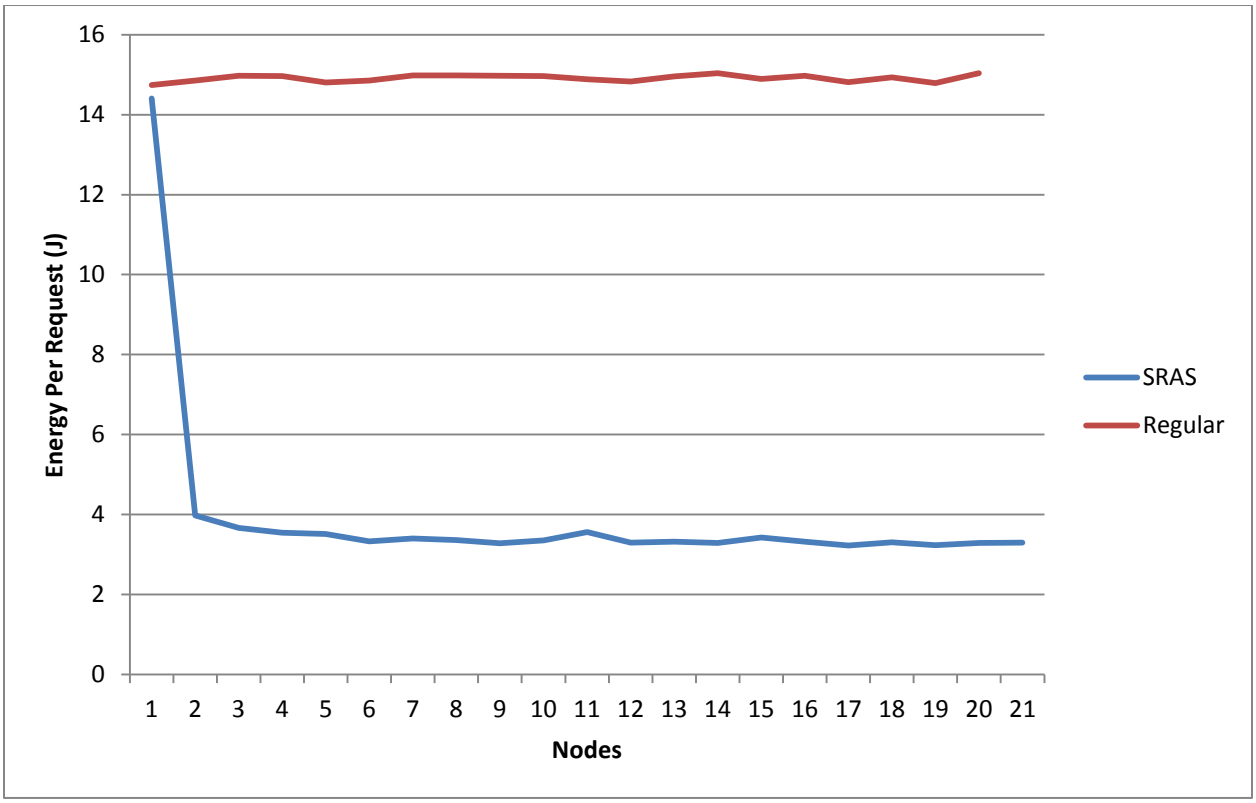


Figure 13 Average energy consumed per request at the cluster nodes.

In figure 13, the average energy per request for the cluster nodes is shown as more CNs are added to the cluster. It is compared against the average energy per request for when instead the nodes make

requests over their own 3G interfaces. In this case, once more than a single CN is in the cluster, the CNs can safely offload all of their requests to the CH (since the queue length never becomes to large). Therefore, the CNs never have to use their own 3G interfaces, and the only energy used is for WiFi interface maintenance and WiFi data transfer. At 2 nodes, the average energy used for a request is 3.98 Joules, and the average decreases by about 17% to 3.3 Joules per request when there are 20 CNs in the cluster. This decrease can be attributed to the decrease in delay per request that is experienced when additional nodes are added to the cluster. Because it takes less time to deliver a data response, the WiFi interface is maintained for a shorter amount of time, leading to a small reduction in energy usage. Similarly, once a CH is introduced, SRAS always uses significantly less energy than if it were not used – in this case, between a 73% to 78% energy reduction over the regular 3G energy consumption.

Table 2 Average energy per request when varying the 3G rate and data size.

| Average Energy Per Request (J) | | | |
|---|---|---|---|
| | 1 KB | 10 KB | 100 KB |
| 1 Mbps | 3.33 | 3.45 | 4.01 |
| 2 Mbps | 3.40 | 3.44 | 3.93 |
| 3 Mbps | 3.32 | 3.53 | 3.99 |

Table 2 shows how data size and 3G data rate impacts the average energy per request. Since all of the requests are being offloaded to the CH, the source of this energy consumption is the CN maintaining its WiFi interface, waiting for the data reply to arrive from the CH. Therefore, with larger data, it will take longer for the CH to retrieve the data, so the CN must keep its WiFi interface on longer while waiting for the reply.

### 3.3.2 Clusterhead

In the same tests as held in the previous section, the delay and energy measurements of the CH were collected separately. Since the CH performs the requests on behalf of the CNs, only a fraction of the requests it performs are for it exclusively. Because it is performing more requests, and at a high frequency, its own requests should benefit from less energy consumption (since it requires less 3G state promotions and less unnecessary tail energy) and smaller delay (since there are fewer 3G connection delays incurred) as the number of CNs increases. For the following results, the energy consumption and delays for the CH's own (self-initiated) requests are analyzed. Figure 14 shows how the delay of the CH's requests varies as the number of CNs is increased.

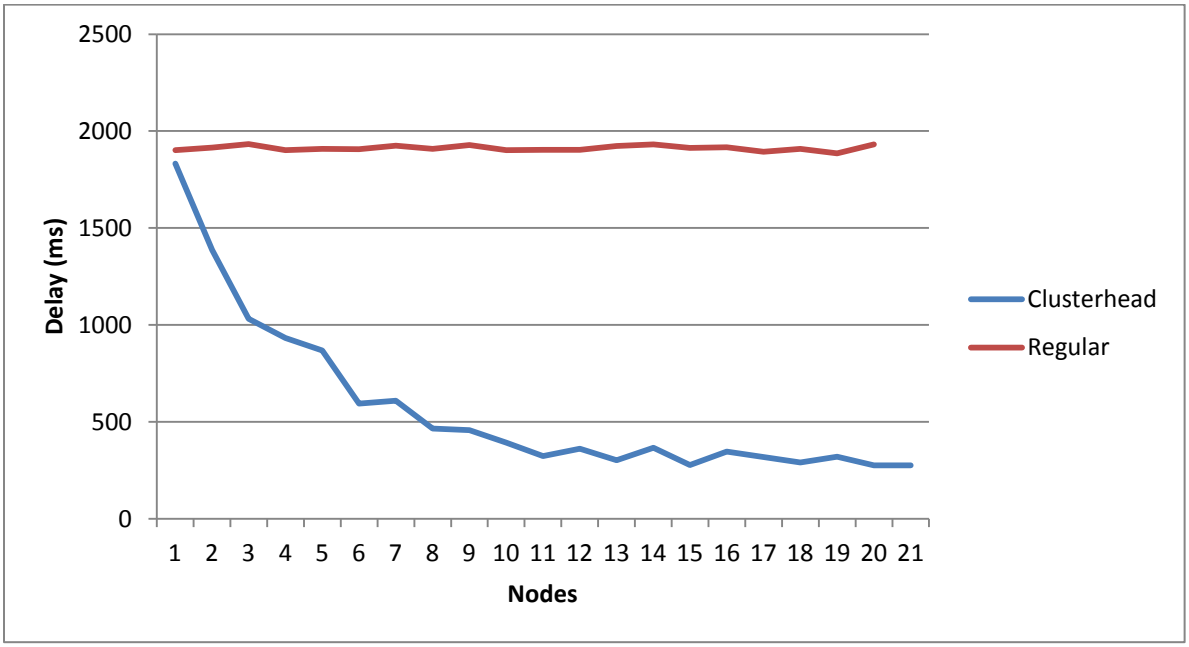


Figure 14 Average delay per request at the clusterhead

In figure 14, the increase in the number of CNs leads to a dramatic decrease in the average request delay for the CH's requests. As expected, the CH's own requests benefit even more from the CNs than vice versa. While both the CNs and the CH benefit from the CH's 3G state and rates, the CH's own

requests do not suffer from the initial delay of the ping request/response, or from having to transmit the reply data back to the requesting node, since the CH's own requests originate locally. With this particular 3G rate of 3 Mbps and 50 KB requests, the CH's request delay improves 85% over not using SRAS. The delay for the regular access scheme again shows that, with the increase in nodes, the access delay does not improve since each node is using its own 3G interface independently.

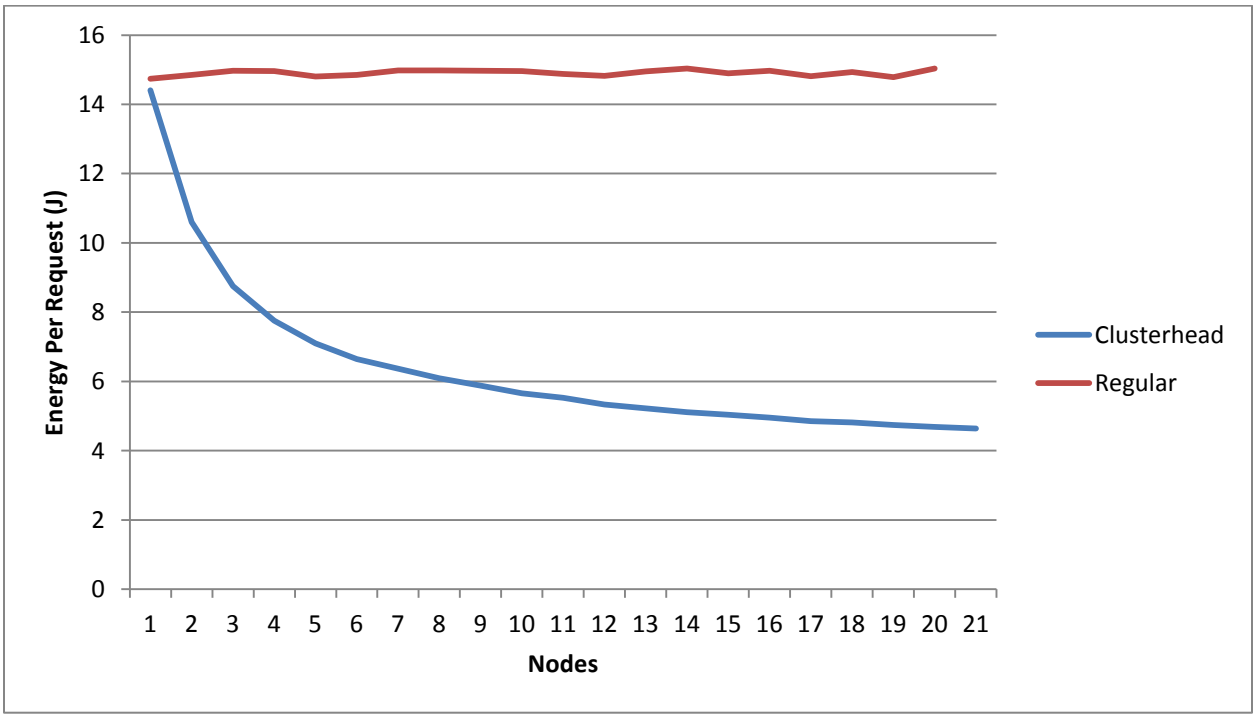


Figure 15 Average energy consumed per request at the clusterhead

The average energy consumption per request for the clusterhead's requests is shown in figure 15. While the CH will use more total energy since it is performing many more requests, the energy per request becomes very low as the number of CNs increases. The energy per request measurement is simply the total energy used divided over the total number of requests (both its own and the requests it performs for the other nodes). As the number of nodes increases, the frequency of request arrivals increases. In turn, this causes the CH's interface to remain in DCH and be able to perform requests without a

connection delay. And, if the node remains in DCH indefinitely, there is no tail energy. For even 1 cluster node, the energy efficiency of the CH improves 25% because it is able to benefit from less 3G state promotions and less tail energy. On average, the CH does use more energy per request than the CNs: 6.38 J compared with 3.92 J on average for the CNs. However, this is because the vast majority of the time the CNs are not using 3G at all. Still, both use only a small fraction of the greater than 14 J average used per request when each node requests data without SRAS.

Table 3 Average clusterhead energy per request when the 3G rate and data size vary.

| Average Energy Per Request (J) | | | |
|---|---|---|---|
| | 1KB | 10KB | 100KB |
| 1 Mbps | 5.82 | 6.12 | 8.92 |
| 2 Mbps | 5.81 | 6.06 | 8.85 |
| 3 Mbps | 5.81 | 6.05 | 8.83 |

Table 3 shows the average energy consumed per request as the 3G rate and data sizes vary. This data demonstrates that the energy used varies more with the size of the data, while increasing the rate only marginally improves the energy use. When the data size increases, both the time spent using 3G increases as well as the time and energy spent transmitting the request over WiFi.

# Chapter 4

# Conclusion

Increasingly, many interesting and important mobile applications require an active connection to the internet. In many cases, smartphones can only access data over the internet by using their 3G interfaces, especially when no WiFi infrastructure exists in the user's location. However, this can be cost prohibitive for some users. In addition, infrequent internet usage over 3G can incur significant delay and energy overheads. To address these challenges, we can develop hybrid data access schemes that allow smartphone users to access internet data without an active 3G connection, or even aggregate requests to reduce access delays and use less energy.

In part 1, the first design and implementation was presented for cooperative caching on Android-based smartphones, using the cooperative caching schemes presented in [14]. The proposed system is based upon the work done in [15], and builds upon the passive protocol and asymmetric design presented by Cao et al. It is implemented as an Android Service, and uses 802.11 ad hoc WiFi to facilitate the ad hoc networking. By using an ad hoc network to access data through a nearby 3G-enabled smartphone, users can still access the internet without a 3G data plan. By using cooperative caching, popular data items are cached throughout this ad hoc network and the delay to access the data is significantly reduced. In the test bed evaluation, it was shown that access delays can be reduced by 25% to 30%, while not requiring large cache sizes.

In part 2, a hybrid scheme for request aggregation in a cluster of smartphones was presented. In this scheme, the cluster nodes intelligently forward requests for data to a clusterhead that performs the requests over 3G on their behalf. As the number of cluster nodes increases, the average access delay and energy consumption per request is reduced significantly. While the clusterhead incurs higher total energy consumption, the delay and energy consumptions for its own requests are significantly reduced because it

can benefit from fewer 3G interface state promotions and fewer connection delays. Through simulation, the hybrid scheme was shown to significantly reduce delays and energy consumption over the regular access scheme.

## 4.1 Future Work

In the cooperative caching scheme for part 1, we assume that the nodes in the network have knowledge of the location of the nearest 3G node. To improve this work, we should take into account node mobility, and determine if another nearby 3G node must be found if the current 3G node has moved or turned off 3G. In addition, an appropriate incentive model could be developed to help motivate 3G users to offer their service to the other non-3G devices, since the 3G device will use more energy and bandwidth by fulfilling the other node's requests. Finally, additional interfaces could be explored, such as Bluetooth, to offer a lower power ad hoc solution.

In part 2, there are a few conditions that could negatively affect how the clusterhead performs. First, if there are too many nodes in the cluster, or if the frequency of cluster requests to the clusterhead is too large, then the clusterhead's request queue may become large and the clusterhead could become a bottleneck. This problem can be alleviated by choosing another node to act as the clusterhead to forward future requests to. Similarly, another potential problem is the large amount of energy required of the clusterhead when there are many requests and it operates over a long period of time. The clusterhead should be rotated based upon some scheme, for example by rotating to the node with the largest amount of battery life remaining. In addition, this will require some additional messaging or a modification of the current messaging to support. For example, the current clusterhead may ping the other cluster nodes to find the device with the largest remaining energy. Then, the current clusterhead may select this new node as the new clusterhead. Similarly, future ping requests to the old clusterhead can inform the cluster nodes of the new clusterhead, without adding significant additional messaging overhead.

In addition, more work can be done to determine the best way to schedule the *ping request*. Instead of sending it every time before the cluster node wants to request data, it can be sent regularly. This rate can be adjusted depending upon the frequency of requests, conditions of the 3G coverage, and node mobility. If it is sent regularly, then the cluster node does not need to incur any delay before deciding whether or not to send its request to the clusterhead, since it can use the information from the previous *ping reply* to make the decision.

# References


[1] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "SoundSense: scalable sound sensing for people-centric applications on mobile phones," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, 2009, pp. 165–178.

[2] E. Koukoumidis, L.-S. Peh, and M. R. Martonosi, "SignalGuru: leveraging mobile phones for collaborative traffic signal schedule advisory," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011, pp. 127–140.

[3] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 77–90.

[4] J. Newman, "Peak Battery: Why Smartphone Battery Life Still Stinks, and Will for Years," *Time*.

[5] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 105–114.

[6] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012, pp. 317–328.

[7] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012, pp. 29–42.

[8] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[9] S. I. G. Bluetooth, "Bluetooth Specification," *Audio Video Wg Adopt. Specif. Revis. V*, vol. 12, 2007.

[10] A. Rahmati and L. Zhong, "Context-for-wireless: context-sensitive energy-efficient wireless data transfer," in *Proceedings of the 5th international conference on Mobile systems, applications and services*, 2007, pp. 165–178.

[11] E. Sarigöl, O. Riva, P. Stuedi, and G. Alonso, "Enabling social networking in ad hoc networks of mobile phones," *Proc. Vldb Endow.*, vol. 2, no. 2, pp. 1634–1637, 2009.

[12] S. Nittel, M. Duckham, and L. Kulik, "Information dissemination in mobile ad-hoc geosensor networks," in *Geographic information science*, Springer, 2004, pp. 206–222.

[13] W. Gao and G. Cao, "On exploiting transient contact patterns for data forwarding in delay tolerant networks," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, 2010, pp. 193–202.

[14] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *Mob. Comput. Ieee Trans.*, vol. 5, no. 1, pp. 77–89, 2006.

[15] J. Zhao, P. Zhang, G. Cao, and C. R. Das, "Cooperative Caching in Wireless P2P Networks: Design, Implementation, and Evaluation," *Ieee Trans. Parallel Distrib. Syst.*, vol. 21, no. 2, pp. 229–241, Feb. 2010.

[16] "Android and iOS Combine for 92.3% of All Smartphone Operating System Shipments in the First Quarter While Windows Phone Leapfrogs BlackBerry, According to IDC" [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS24108913.

[17] "The Bump App for iPhone and Android | Bump Technologies, Inc." [Online]. Available: http://bu.mp/company/faq.

[18] S. Trifunovic, B. Distl, D. Schatzmann, and F. Legendre, "Wifi-opp: ad-hoc-less opportunistic networking," in *Proceedings of the 6th ACM workshop on Challenged networks*, 2011, pp. 37–42.

[19] "Service | Android Developers." [Online]. Available: http://developer.android.com/reference/android/app/Service.html.
[20] "ContentProvider | Android Developers." [Online]. Available: http://developer.android.com/reference/android/content/ContentProvider.html.
[21] "adhoc-on-android - Google Project Hosting." [Online]. Available: https://code.google.com/p/adhoc-on-android/.
[22] "Intent | Android Developers." [Online]. Available: http://developer.android.com/reference/android/content/Intent.html.
[23] "Uri | Android Developers." [Online]. Available: http://developer.android.com/reference/android/net/Uri.html.
[24] "BroadcastReceiver | Android Developers." [Online]. Available: http://developer.android.com/reference/android/content/BroadcastReceiver.html.
[25] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "TOP: Tail optimization protocol for cellular radio resource allocation," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, 2010, pp. 285–294.
[26] *Configuration of fast dormancy*, Std., Rev. Rel. 8. [Online]. Available: http://www.3gpp.org
[27] 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*, 2009, pp. 280–293.

# ACADEMIC VITA

Max Freilich
maxfreilich@gmail.com

---

**Education**

B.S., Computer Science, 2013, Penn State University, University Park, PA

M.S., Computer Science and Engineering, 2013, Penn State University, University Park, PA

**Professional Experience**

Nicira Networks, Intern, Summer 2012

Cisco Systems, Intern, Summer 2010 – Summer 2011

MLB Advanced Media, Intern, Summer 2009

**Honors and Awards**

Schreyer Honors College Integrated Undergraduate/Graduate Program

Dean's List for Academic Achievement (Fall 2008 – Fall 2012)

Schreyer Honors College Academic Excellence Scholarship (Fall 2008 – Spring 2012)

Penn State President's Freshman Award

1st place, Penn State Cisco Innovation Challenge