University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Energy-efficiency in Mobile DHTs

Nico Rutishauser
Zürich, Switzerland
Student ID: 09-706-821

Supervisor: Thomas Bocek, Daniel Dönni
Date of Submission: March 5, 2015

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

The performance gap between mobile phones and computers shrinks with advancing mobile phone technology. Powerful mobile devices with fast networking can easily participate in an overlay network like a Distributed Hash Table. But NAT traversal remains a challenge of P2P systems because peers need to be able to interact with each other. Another challenge is the limited power capacity of mobile devices. Networking is one of the most battery-draining component of mobile phones, but essential for the participation in a P2P network. This thesis proposes multiple NAT traversal techniques using relay peers and long-living TCP connections between the relays and the mobile phones. To reduce the battery consumption, buffered relaying techniques are presented that allow the mobile phone to longer remain in the sleep mode. Another optimization is the replacement of the long-living TCP connection with the mobile operating system's push notification service. The evaluation confirms a reduction of the energy consumption when using the optimized buffered push notification relaying technique. While only 6% of energy can be saved in Wifi, up to 30% less battery is consumed in a 3G cellular network. These insights are used for the implementation of a distributed file synchronization client on Android.

Zusammenfassung

Der Leistungsunterschied zwischen Mobiltelefonen und Computern sinkt mit laufendem Technologiefortschritt. Mobiltelefone mit einer schnellen Internetverbindung können problemlos an einem P2P Netzwerk teilnehmen. NAT Traversal ist eine Hauptschwierigkeit in P2P Systemen, da die Peers direkt miteinander kommunizieren müssen. Die limitierte Batteriekapazität von Mobiltelefonen ist eine zweite Herausforderung, denn die Verwendung von mobilen Daten ist sehr batterieintensiv, jedoch unverzichtbar für die Teilnahme an einem P2P Netzwerk. In dieser Arbeit werden verschiedene NAT Traversal Techniken mittels Relaypeers und langlebigen TCP-Verbindungen zwischen den Relays und Mobiltelefonen aufgezeigt. Es werden auch gebufferte Relaytechniken präsentiert, welche es dem Mobiltelefon ermöglichen, länger im Standby-Modus zu bleiben, um so den Batterieverbrauch zu reduzieren. Eine weitere Optimierung ist die Verwendung der im mobilen Betriebssystem eingebauten Push-Funktionalität anstelle einer langlebigen TCP-Verbindung. Die Evaluation bestätigt bei Gebrauch einer gebufferten und optimierten Relaytechnik eine Energiereduktion. Während im WLAN lediglich 6% der Energie eingespart werden kann, verbraucht diese Technik in einem 3G Netzwerk bis zu 30% weniger Strom. Die Erkenntnisse werden bei der Entwicklung einer Android-App für die verteilte Dateisynchronisierung eingesetzt.

Acknowledgements

For the support of this master thesis, I want to thank several people. First, I would like to express my special appreciation and thanks to Dr. Thomas Bocek for supervising me during this master thesis with patience, passion and enthusiasm. I also like to thank Fabian Kaup and Julius Rückert for the help and assistance with the energy measurements. Special appreciation also to Prof. Dr. Burkhard Stiller for the opportunity to write this thesis at the Communication Systems Group. Last but not least, I am grateful to my family and friends for their ongoing support and motivation.

Contents

Chapter 1

Introduction

1.1 Motivation

With the progress of mobile phone techonology, new research opportunities arise. The evolution of the cell phone goes from bulky devices for calls only to light and fast smartphones. While the calling functionality becomes just a subordinate feature, personalizing the phone with applications for one's own needs has become the focal point. This change is enabled by the steady improvement of the network bandwidth and latency with every mobile network generation [20].

Another trend is the distribution of tasks onto multiple nodes. Examples of why tasks are being distributed on multiple nodes are scalability, stability and security [21]. One of the most common P2P structure that benefits from these properties is the Distributed Hash Table (DHT). Key-Value pairs can be stored and retrieved to and from participating and self-organized peers.

DHTs are the foundation of P2P file-sharing, which is responsible for a large portion of the internet traffic around the world [4]. While the total traffic of file-sharing among fixed-network peers will remain constant, it is expected that the traffic in mobile networks exhibits a compound annual growth rate (CAGR) of 26% [11]. The combination of mobile phones and DHTs therefore is a promising research field.

1.2 Description of Work

The aim of this thesis is to enable and optimize DHTs on mobile phones. Differences between fixed networks and mobile networks cause two major challenges. Firstly, the NAT and firewall of the mobile network is not in control of the user but managed by the mobile network operator. A peer in a DHT needs to be reachable such that other peers can store and retrieve content or ask for routes. The second challenge is the limited battery capacity of mobile devices. An active wireless network adapter is, apart from the display, the most energy-consuming component of a mobile device [24]. In order to maintain the

DHT structure, many control messages are exchanged that keep the network adapter busy. Adaptations of the P2P protocol are proposed, allowing the network adapter to remain on standby as long as possible. The main contributions of this thesis are:

- Achieve NAT traversal by adding relay peers to the DHT that help to send requests through the NAT and firewall devices to the mobile phone (section 3.1).
- Buffer requests before transmitting them to the mobile device, such that multiple network tasks can be performed simultaneously (section 3.1.3).
- Use the shared push notification service of the mobile operating system to reduce the energy consumption at the mobile phone (section 3.2).

In particular, this thesis investigates the impact of the protocol adaptations on the mobile phone's energy consumption. Can the addition of a buffer and the use of the shared push notification service reduce the battery drain of the mobile phone?

The proposed changes on the P2P protocol are added to an existing open-source DHT implementation. It will be optimized for Android devices and their effects on the mobile phone's energy consumption will be measured. As a proof of concept, a secure P2P file sharing application for the Android platform is developed using the optimized DHT library (section 4).

The evaluation compares three different NAT traversal approaches in various connection settings within a small-scale P2P network (section 5). The total energy consumption of the mobile phone is measured using a data acquisition device. In addition, a process on the mobile phone estimates the energy consumption of the network adapters by monitoring their usage.

Chapter 2

Related Work

To get an overview of the current state of research, this chapter presents a brief description of the related work and of its correlation to this thesis. First, the power states of the wireless networks adapters are analyzed. An overview of recent experiments on energy measurements and their applications in the context of P2P systems are presented next. This section ends with a brief summary of how the energy consumption of smartphones can be quantified.

2.1 Wireless Network Adapter States

The energy consumption of wireless network adapters fluctuates significantly. Feeney and Nilsson [3] show that the energy consumption does not correlate with the bandwidth utilization. This is because the wireless network adapter has different states.

2.1.1 3G Power States

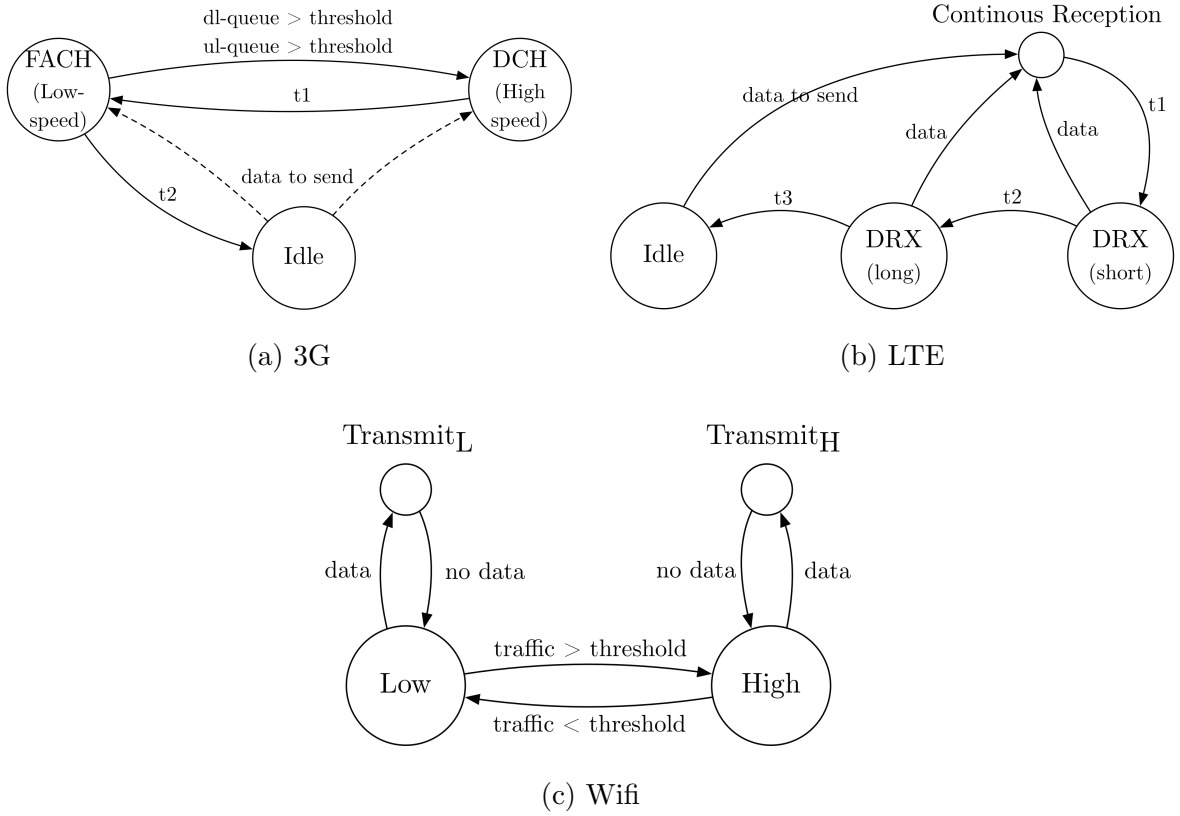
Figure 2.1a shows the three states of the 3G network adapter [5][18][19]:

IDLE: The IDLE mode consumes almost no energy and is the standby state in case no network connection is used.

DCH: In the DCH mode, the full network speed is available resulting in the highest energy consumption. The absolute energy consumption depends on the network carrier and the mobile device, but typically is between 600mW and 800mW.

FACH: In contrast to the DCH mode, FACH does not provide a dedicated connection but reaches connection speeds of a few kb/s only, which may be enough for small data transfers. In this state, the network adapter consumes approximately 50% less energy than in the DCH state.

As soon as data needs to be sent, either the FACH or the DCH mode is activated. The selection of the mode depends on the network carrier [19]. If FACH is activated and the queue to send data reaches a certain threshold, the network adapter promotes from FACH to the faster DCH mode, where the full network speed is available. Due to the long signaling distance between the mobile phone and the base station, the transition from IDLE to FACH or DCH mode takes up to two seconds. To prevent repetitive activation and deactivation and to reduce the total delay during recurring network activity, the network adapter remains in the DCH or FACH mode until the respective inactivity timeouts t_1 or t_2 (also called tail-times) occur.



(a) 3G

(b) LTE

(c) Wifi

Figure 2.1: Network adapter power states

2.1.2 LTE Power States

In 2004, Long Term Evolution (LTE) has been proposed as the fourth generation mobile network. Higher down- and uplink rates and lower latency are the main improvements compared to 3G. A simplified power state model can be found in Figure 2.1b. If there is no data to transfer, the LTE adapter remains in the IDLE state. In this state, the network adapter is in a sleep-cycle, waking up about every second to check if data transmission is required. If data is ready for transmission, the state is promoted to the high energy consuming Continuous Reception state. Timeouts t_1, t_2 and t_3 prevent the network adapter from demoting back to the IDLE state too fast. The energy consumption in DRX_{short} and DRX_{long} is lower than in Continuous Reception, because of micro-sleeps,

similar to the IDLE state. DRX_{short} and DRX_{long} solely differ in the sleep cycle duration. While t_1 and t_2 are just a few milliseconds long (usually 20ms and 40ms), inactivity timer t_3 is more than 10s [10].

2.1.3 Wifi Power States

The Wifi adapter power model has four states, which are shown in Figure 2.1c. The two transmit states $transmit_L$ and $transmit_H$ are only active approximately 10-15 milliseconds per second, then demotes back to either LOW or HIGH [18]. Higher bandwidth and shorter tail times reduce the total energy consumption in typical scenarios compared to 3G and LTE. Instant power state transitions are possible in Wifi because distances are much smaller than in cellular networks and faster signaling is possible.

European regulations allow a maximum equivalent isotropically radiated power (EIRP) of 100 mW in 2.4 GHz Wifi networks. In contrast, the maximum transmission power of a mobile phone in 3G networks is up to 2 W (power class 1). More power is necessary because the transmission distance between the mobile phone and the base station is higher than for Wifi.

2.2 Energy Consumption in Mobile P2P

Kelényi et al. [14] compare the energy consumption of a mobile device when acting as a *full-peer* and when running in *client-mode*. A peer running in full-mode is reachable through a public IP address and answers all requests. In the client mode, requests from other peers are dropped in order to reduce the load at the mobile peer. The peer only performs routing table updates every 15 minutes. In their experiment, the full-peer mode answers 36 times more messages and has an increased energy consumption of factor six compared to the client-mode.

With increasing number of client-mode peers, the DHT becomes instable because only full peers can conduct message routing. As a countermeasure, Kelényi et al. [15] take the idea one step further. They not only differentiate between full-peer mode and client-mode, but propose probabilistic message dropping. The lower the battery state of the mobile peer, the more messages are dropped and thus the less energy is consumed. Furthermore, they show that in a DHT implementation with parallel routing (as this is the case in Kademlia [17]), the routing latency is not significantly influenced as long as there are more than 70% of fixed network peers in the DHT.

Compared to this thesis, only experiments in Wifi with fixed public IP addresses and opened firewalls are carried out, but not in cellular networks [14][15]. Measurements with client-mode peers connected to cellular networks are conducted by Kassinen et al. [12]. The latter varies the number of peers in the DHT, churn rate and the lookup interval (making a *get* request). Every combination of these parameters leads to a similar total energy consumption. Another experiment in the study is non-P2P-related and evaluates the energy consumption of the mobile network adapter with numerous combinations of

the packet data size and the intervals at which the packets arrive. In regard of the state diagram in Figure 2.1a, a clear drop of total energy consumption can be observed as soon as the data becomes small enough and/or the interval large enough such that the network adapter can switch from DCH to FACH or even remain in the FACH state.

2.3 Mobile DHT Applications

There are many BitTorrent clients for Android[1] and iOS[2] available, but all of them are either running in client-mode or in the very battery draining full-peer mode. The reason is that the BitTorrent protocol is given and every peer needs to be in compliance with it. When changing the protocol (as for example in client-mode), backward compatibility must always be maintained. Nurminen et al. [18] compare the energy consumption of downloading a file over HTTP to downloading it over a mobile BitTorrent client. The P2P download requires only 20% more energy compared to downloading it over traditional HTTP in a 3G network. This overhead can be explained by the longer download time due to the additional complexity of the BitTorrent protocol. The study shows that P2P file sharing on mobile devices, as it is also shown in this thesis, is feasible.

Because of large data transfers in online video streaming, P2P becomes a widely used practice in that area, too. Wichtlhuber et al. [23] created a flexible P2P video streaming framework with interchangeable DHT implementations. Their benchmark platform is open-source but lacks effective measurements.

2.4 Measuring Energy Consumption

Determining the energy consumption of Android devices through software is non-trivial because the operating system does not offer such a service.

Zhang et al. [24] constructed a power estimation model based on the current usage statistics. The model is device-dependent and its accuracy is within 5% of the real energy consumption. Only models for outdated phones like the HTC Dream (2008) or HTC Sapphire (2009) are available, but they can be applied on any device for comparison reasons using the available PowerTutor[3] application.

The advantage of software measurements over hardware measurements is that no special hardware is required and fast feedback is possible. Moreover, it is possible to estimate the energy consumption of one specific running process. In contrast, hardware measurements can only show the overall energy consumption of a mobile phone. Although hardware measurements are more exact, results may be biased or disturbed by background processes. A measurement testbed for USB powered devices has been designed by Gross et al.[8] and Kaup et al. [13]. A similar setup will be used for carrying out the evaluation in this thesis.

[1] `http://www.android.com/`

[2] `https://www.apple.com/ios/`

[3] `http://ziyang.eecs.umich.edu/projects/powertutor/`

Chapter 3

Solution Design

This chapter first introduces the NAT traversal problem with P2P nodes on mobile phones. As a resolution, multiple relaying techniques are proposed. Next, these relaying techniques are enhanced step-wise to improve their energy-efficiency in order to reach a longer battery lifetime of the mobile phone. At the end of this chapter, a relaying technique that uses a combination of the push and pull strategy is presented.

3.1 NAT Traversal

A Network Address Translation device (NAT) is the boundary between a private network and the Internet. Its main task is the translation of outgoing and incoming requests to the appropriate IP address space. The NAT's public IP is set as the source address in outgoing requests and the real requester is memorized in order to route the according response back to it. An incoming message that is not linked to any request cannot be forwarded to the correct receiver and is dropped [22]. In addition, NAT devices usually contain a firewall protecting against intrusion into the private network. They monitor the traffic and block it if certain rules apply or an abnormality is detected. Unreachability is a problem in decentralized P2P systems because peers need to interact and be able to initiate a communication with any other participating peer. A peer running on a mobile phone is typically either behind the NAT of the mobile network operator (cellular) or behind the NAT of the Wifi provider. To overcome this problem, there are various NAT traversal techniques such as *Universal Plug and Play* (UPnP), *Application Level Gateways* (ALG) or *Hole Punching* [9]. In this thesis, NAT traversal is solved by using well-reachable relay peers. It is applicable because relaying is completely independent of the NAT capability and the network setup. It also works when the peer is behind multiple (layered) NAT devices.

The following terminology is used: A *mobile peer* is a peer running on a mobile phone and is unreachable because it is always behind a NAT. It is therefore also labelled as an *unreachable peer*.

3.1.1 Relay Peers

A relay peer acts as a proxy for an unreachable peer. The unreachable peer initiates a duplex and long-living TCP channel to one or multiple relay peers. A message to an unreachable peer is routed to one of its relay peers which then forwards it through this open channel to the true recipient. The NAT does not block these messages as it associates them with the open TCP channel. Since most NAT devices have a timeout on these open connections, regular heartbeats from the unreachable peer to the relay peer are necessary to refresh the mapping at the NAT.

With the introduction of relay peers, the P2P network becomes heterogenous. Some well-reachable nodes have a special role in serving unreachable peers. Peers with special roles or additional server-like functionality are often denoted as *super-peers*. It has been shown that the overall performance increases when powerful peers take the lead as super-peers and weak nodes interact with those super-peers only [1]. If weak peers would also be responsible for routing and other P2P maintenance tasks, performance would decrease.

Applying the concept to mobile DHT, mobile devices are weak peers and fixed-networked relay peers can be seen as super-peers. To prevent single-points of failure, mobile peers can connect to multiple relays. A message to an unreachable peer can be sent to any of those relay peers to get forwarded to the mobile peer. Request that originate from the mobile peer are not sent over the relay peer but made directly, since this is a common request/response scenario for the NAT.

3.1.2 Control Messages

Besides forwarding requests, the duty of a relay peer can be expanded by answering DHT control messages on behalf of the connected unreachable peers. In general, there are two types of control messages in DHTs:

Aliveness check: Every peer maintains a unique routing table. Its entries are pinged regularly in order to remove peers that left the network.

Routing: A peer that wants to store or retrieve data in the DHT needs to find the node responsible for the given key. Since the unique routing table only holds an extract of all peers in the DHT, peers needs to ask neighbors for the address of the responsible node. The routing table is denser for close (logical) neighbors and contains only few entries of peers at the other side of the DHT. Iteratively, the target peer is found by progressively asking closer peers.

To check whether the steady TCP connection between the relay peer and the unreachable peer is still alive, heartbeat messages are exchanged. While the connection remains open, the relay can assume that the unreachable peer is alive and aliveness checks from other peers in the overlay network can be confirmed. If the connection is closed, the unreachable peer might have left the DHT or disconnected due to network problems. In both cases, the unreachable peer can be regarded as *offline* and the aliveness checks are declined.

Routing requests are more complicated to answer because the relay peer needs to know the entries in the routing table of the unreachable peer. The relay peer cannot answer routing requests with its own routing table since the location of the unreachable peer in the DHT could differ completely. Therefore the routing table entries are regularly sent to the relay peers as piggy-back data of the heartbeat. The necessary interval depends on the DHT activity and the churn rate.

3.1.3 Relaying Mechanisms

In the following section, three kinds of relaying are presented: *direct relaying*, *reverse connection relaying* and *buffered direct relaying*. It is not the relay peer that decides which mechanism an unreachable peer uses, but the unreachable peer itself.

Direct Relaying

Figure 3.1 shows an example of direct relaying. The request to an unreachable peer is sent to one of its relay peers, which then forwards it through the open TCP channel. This example assumes that the requester is publicly reachable in the P2P network, otherwise, the response (3) would also need to be sent via a relay peer. The mechanism requires a minimum number of messages, but has two major drawbacks. First, the end-to-end notion is lost. The relay peer can read and modify the content of the message or even drop it. If no end-to-end encryption is implemented, the requester and the unreachable peer need to trust in their relays. Secondly, transferring the message to the relay and then to the unreachable peer can be inefficient. Consider the following example: Both the requesting peer and the unreachable peer are situated in Europe while the relay peer is located in Australia. A message travels from Europe to Australia and back to Europe.

Reverse Connection Relaying

Reverse connection relaying is more complex and requires two TCP connections, but the disadvantages of direct relaying are eliminated. Figure 3.2 shows an example situation for reverse connection relaying. Instead of sending the request to the relay peer, the requester sends a small *hint message* via the relay peer through the permanently open TCP channel to the unreachable peer. The hint message does not contain any payload, it only serves to notify the unreachable peer about a request. The unreachable peer can obtain the real request from the requester directly and answer as soon as it has been processed. This mechanism is called reverse connection because the unreachable peer opens a dedicated channel to the requester in order to receive the message and respond to it. Optionally, the channel can be kept open for further data exchange. While the end-to-end notion is preserved, reverse connection relaying is only applicable if the requester is publicly reachable. Otherwise, the reverse connection cannot be established. Therefore, direct relaying needs to be the used if two unreachable peers need to interact.

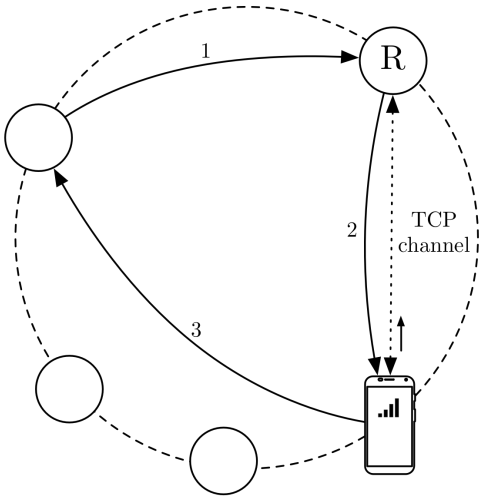


Figure 3.1: Direct relaying: The request is sent to the relay peer R (1) and immediately transferred to the mobile phone through the open TCP channel (2). The mobile phone replies directly to the requester (3).

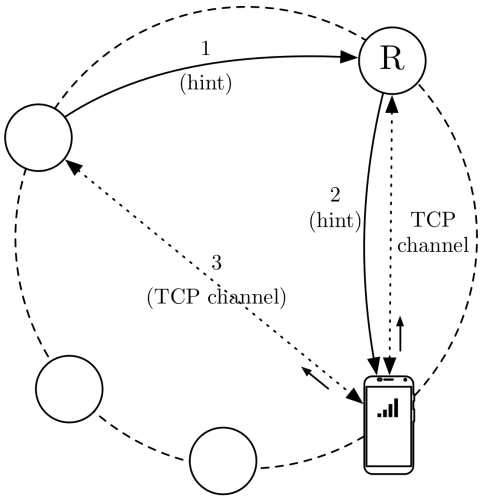


Figure 3.2: Reverse connection relaying: Instead of sending the request, a hint message is sent to the relay peer R (1). The relay notifies the mobile phone (2), which can obtain the request directly from the requester by opening another TCP channel (3).

Buffered Direct Relaying

The direct relaying and reverse connection relaying approach have fast response times because requests are forwarded to the unreachable peer immediately. Peers with unlimited resources (like bandwidth and energy) can always be in attendance and respond requests instantly. In contrast, mobile devices have limited battery power. The continued readiness and maintaining the open channel consumes much more energy because the network adapter can hardly switch to standby mode. The energy consumption of an active network adapter due to high P2P activity is approximately equal to the energy consumption of a voice call [18]. This is a convenient metric because most mobile phone vendors provide the maximum talk time to full discharge. Measurements show a battery life of a mobile phone in full-peer mode of 3 hours in UMTS and 5-10 hours in Wifi [12]. A mobile application with such high energy consumption is infeasible because battery-intensive applications are often given poor ratings and evaded by end-users [16].

The high energy consumption can be eliminated at the cost of slower response times. Figure 3.3 shows an example of *buffered direct relaying*. Instead of notifying the unreachable mobile peer at every request, the relay peer first collects multiple requests in a buffer. Buffering and sending messages in bursts is a common practice to improve efficiency. A longer interval between notifications from the relay peer to the unreachable peer allows the mobile phone's network adapter to sleep and save energy. When the buffer is transferred to the mobile phone, the wireless network adapter wakes up and switches temporarily to FACH state (or Continuous Reception in LTE or $\text{transmit}_{\text{H}}$ in Wifi) to process all requests at once. This behavior consumes less battery than permanently remaining in

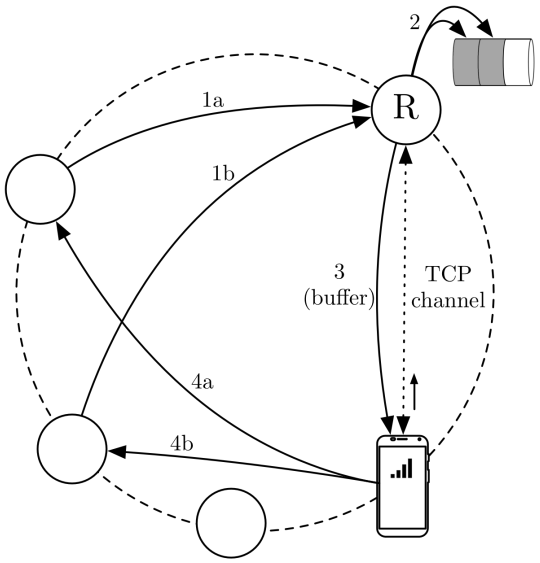


Figure 3.3: Example for *buffered direct relaying*. Two requests (1a and 1b) are buffered at the relay peer R (2) until a certain timeout occurs or the buffer is full. The buffer is then sent to the mobile phone through the open TCP channel (3). The mobile phone can process the requests and respond them (4a and 4b).

(low-powered) transmission mode.

The requester is notified about the existence of the buffer and can decide whether to wait for the response of the unreachable device or to drop the request. In DHT implementations with replication, data could also be obtained from a replica node.

Buffer Limits

As the first request for the unreachable peer is added to the buffer, a timeout starts to count down. It triggers, as soon as the oldest (first) message in the buffer reaches the given age. This ensures that a message never resides longer in the buffer than the configured timeout. A second, non-exclusive limit is the buffer size. Before a buffer overflow happens, the relay peer notifies the unreachable peer to download the buffer. As soon as either the size or the age of the buffer reaches the limit, the notification is triggered.

3.2 Push Notifications

Push notifications are used by many mobile applications like messengers, email clients or news portals. The *push* strategy results not only in a reduced latency compared to the *pull* strategy, but is also more efficient. The channel is only used when messages arrive. With the *pull* strategy, regular polling for updates is necessary.

The permanently open TCP channel can be used to push messages to the unreachable peer. If every application would open and maintain its own long-living TCP channel to receive notifications, the network adapter would always be busy with sending heartbeats or re-opening lost connections. Therefore, most mobile operating systems provide a shared

push notification service[1]. The phone maintains a single open connection to the vendor's push server, highly optimized in terms of energy-consumption and scalability.

3.2.1 Google Cloud Messaging

The official push notification service for Android devices is called *Google Cloud Messaging* (GCM). An application needs to register itself at the Google servers to receive notifications from a trusted sender (identified by a unique sender ID). A device- and application-specific registration ID is returned. The sender needs to know this unique registration ID in order to send push notifications to exactly this device.

Despite being a free service, Google prevents overuse by throttling messages using the *leaky bucket algorithm*. If push notifications are sent at a fast pace, they are held back or dropped. The bucket size depends on the current overall service workload. Experiments have shown that throttling is applied on intervals below 20s. Even for the reception of push notifications over GCM with slower intervals there is no guarantee because it depends on the server load which again depends on the daytime, weekday and many other factors. The lack of a confirmation receipt makes the service even more unpredictable.

The ability to receive GCM messages belongs to the Google Play Services which comes with the Google Play Store[2] and is available for all Android-powered devices. The heartbeat interval for the GCM push notification channel is by default configured to 28 minutes in cellular networks and 15 minutes in Wifi.

3.2.2 Google Cloud Messaging in Mobile DHT

To optimize the energy consumption of mobile peers, the functionality of the relay peer is extended to use the push notification service of the mobile operating system instead of their own TCP channel.

The same example situation as shown in buffered direct relaying (see Figure 3.3) is applied to GCM in Figure 3.4. The combination of reverse connection relaying and push notifications is generally denoted as *buffered push notification relaying* (BPN) in the following. This work is confined to Android devices using GCM, but the concept is also applicable to other mobile operating systems using a similar push notification serivce.

For privacy and efficiency reasons, only a small *tickle message* is sent over GCM. Tickle messages are more efficient because they can be collapsed as they do not contain a payload. If the mobile phone is temporarily offline and multiple push notification messages are sent to the push server, only one of them arrives at the mobile phone. The mobile peer can then query the relay peer directly to obtain the buffered requests.

[1] Android: `https://developer.android.com/google/gcm/`
iOS: `https://developer.apple.com/notifications/`
Windows Phone: `http://msdn.microsoft.com/en-us/library/windows/apps/ff402558`

[2] `https://play.google.com/store`

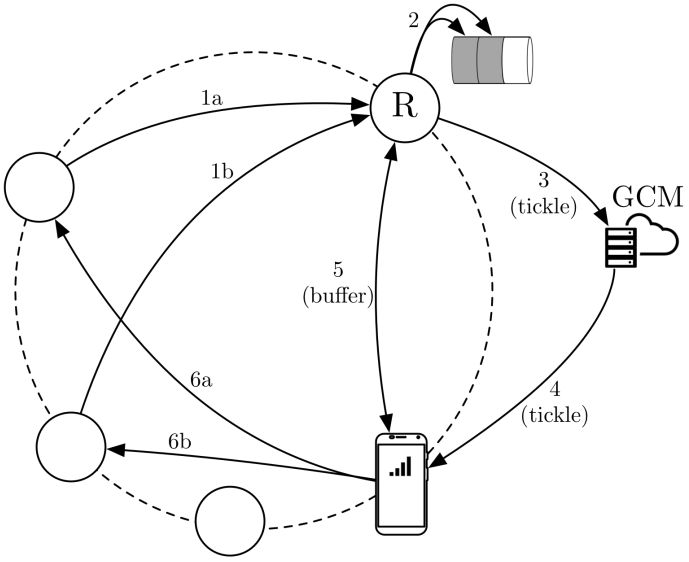


Figure 3.4: This example for *buffered push notification relaying* shows how Google Cloud Messaging in mobile DHT with buffered direct relaying is applied. Two requests (1a and 1b) are sent to the relay peer R which puts them into the buffer. If the buffer is full or times out, the relay peer sends a tickle message to the GCM servers (3). The tickle message is forwarded to the mobile phone (4) as a hint to obtain the buffer from the according relay peer (5). The requests can be processed and replied (6a and 6b).

Sending push messages over GCM requires a personal API key, which needs to be obtained at Google. The distribution of this key to every relay peer in the overlay network could be a security leak. For that reason, the GCM sending mechanism is separated from the relaying functionality. Relay peers without GCM ability can use other peers in the P2P network having the GCM ability to transmit the push message. To prevent throttling at the GCM servers, the relay peer needs to buffer the requests long enough (at least 20s) and then send a single tickle message to the unreachable peer. Multiple relays for the same mobile peer need to share the GCM message rate, thus it is not recommended to strictly use the lower limit.

The advantage of using the mobile operating system push notification implementation is that their own TCP channel can be omitted and the energy consumption can be reduced. The channel from the mobile phone to the push notification servers is in place in any case. Additionally, the push service responsibility is shifted to the Android system, and the separation of concerns in the application is improved. As a drawback, this approach loses the notion of a fully distributed network because of the centralized push servers. In contrast, the buffered direct relaying approach with its own TCP connection is fully distributed but a higher energy consumption is expected. A summary of the presented relaying mechanisms is shown in Table 3.1.

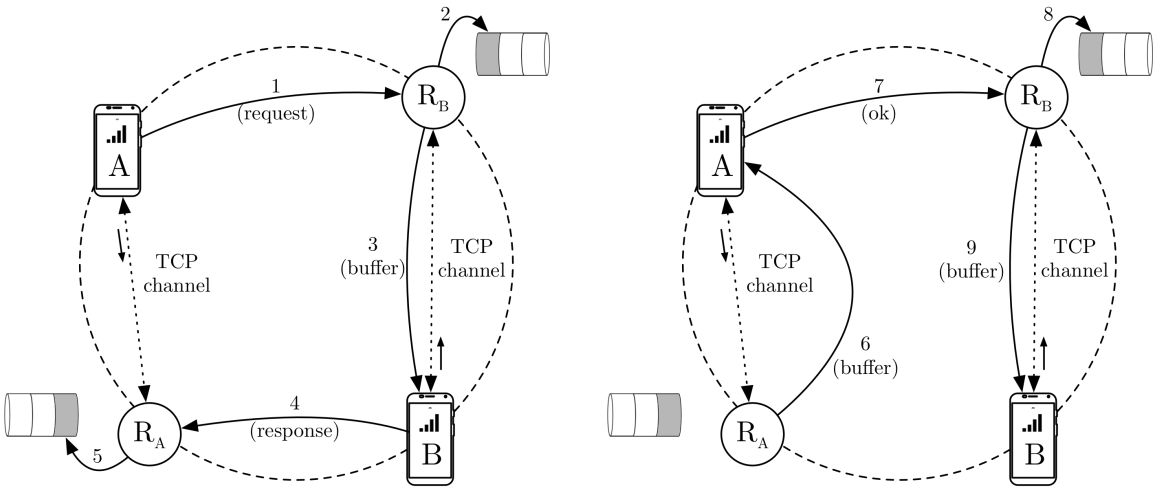
	open TCP channel	push notification
non-buffered	direct relaying reverse connection relaying	not possible with GCM (due to throttling)
buffered	buffered direct relaying	BPN relaying

Table 3.1: A summary of the presented relaying mechanisms, aligned by the buffering and the notification channel used.

Buffered Example with two Mobile Peers

Transfers between two mobile peers becomes more entangled because both peers are relayed. Figure 3.5 shows the procedure, split into two parts for clarity. The mobile peer A makes a request to the mobile peer B. Mobile peer A starts by sending the request to the relay peer R_B (1), the relay peer buffers (2) and notifies (3) the unreachable mobile peer B. After processing the request, the mobile peer B detects that the requester is also relayed and sends the response to one of its relay peers (4), which again buffers it (5). Figure 3.5b shows the continuation of the procedure. As soon as the buffer is timed out or full, the relay R_A notifies the mobile peer A about the response (6). To confirm the reception of the response, the mobile peer A sends a confirmation message again via the relay peer R_B to the mobile peer B (7-9).

The whole sequence includes three buffering iterations until its termination. In the worst case, the latency is a triple of the configured buffer timeout, what enhances the disadvantage of buffered relaying techniques. A direct transfer of the request is not possible because the origin and the target peer are both unreachable. A direct connection could however be established using hole punching [9].



(a) Part I (b) Part II

Figure 3.5: Example with two mobile peers

3.2.3 Polling as Fallback

The unreliability of GCM is often criticized in the Android community. One reason for messages that never arrive (or with high delay) is the already mentioned throttling mechanism against spam and misuse. If a mobile peer is connected to multiple relay peers, the relay peers need to share the message quota for that device. Another reason for push notification failures might be NAT devices that close the open connection faster than the heartbeat interval. The mobile device does not detect connection loss until the

next heartbeat attempt. If the mobile peer does not obtain the buffer in-time, requests time out or a buffer overflow at the relay peer may occur.

As a fallback for the push notifications, the regular routing table update message from the unreachable peer to the relay peer can be (mis)used to transmit the buffer to the mobile phone. The request message contains the current routing table of the mobile peer. The response message does not only contain the confirmation that the routing table update arrived successfully, but also contains currently buffered messages. If the routing table update interval is shorter (e.g. 30s) than the push notification buffer timeout (e.g. 60s), no push notifications need to be sent because the buffer is pre-emptied at a fast pace. However, short routing table update intervals lead to short battery lifetimes. Long intervals (e.g. 2min) are recommended because routing table updates are not time-critical and the buffer appending serves only as a fallback for failed push messages.

Chapter 4

Implementation

This chapter gives insights about the implementation of the proposed relaying techniques. As a proof of concept, an Android client for a distributed file sharing library is designed and implemented. The challenges faced during its implementation are explained next. Finally, screenshots of the Android file sharing client are presented.

4.1 TomP2P

The open and adaptable DHT implementation TomP2P [2] is currently one of the most evolved open-source DHT implementation. Thomas Bocek, founder of the library and supervisor of this thesis, continuously implemented and improved it for more than ten years. It therefore offers a good foundation for the implementation of the proposed relaying approaches.

An existing NAT traversal package already contains a large portion of the direct relaying and reverse connection relaying approach. During this thesis, it was tested, improved and extended by the buffered direct relaying approach. In addition, a new software package for the Android-specific implementation was created. In total, four different handlers exist. During the NAT traversal setup, the unreachable peer specifies the preferred handler.

Direct Relaying: Opens a TCP channel between the unreachable peer and the relay peer. This handler is active by default.

Reverse Connection Relaying: This handler reuses parts of the direct relaying handler for the transmission of the hint message. Additional functionality for opening a reverse connection from the unreachable peer to the requester is required. This handler is also active by default and chosen automatically if two conditions are fulfilled: (1) the requester is publicly reachable and (2) the request message is larger than 1000 bytes.

Buffered direct relaying: Again, the code of direct relaying is used, but requests are first stored in a configurable buffer. The buffer configuration must happen before the registration of the handler and therefore, this relaying technique is not active by default.

Buffered push notification relaying: The buffered push notification relaying resembles the buffered direct relaying but does not open a TCP connection between the unreachable peer and the relay peer. Instead, messages are sent over GCM. Therefore, an API key needs to be in place. If the ability to serve Android devices is required, this handler needs to be configured and registered at the relay peer.

4.2 Implementation of an Android Hive2Hive Client

Hive2Hive is a distributed file synchronization and sharing library [7]. It provides a similar interface as BitTorrent Sync[1], but is open-source and written in the platform-independent language Java. The underlying DHT functionality is provided by the presented TomP2P framework. Hive2Hive provides file handling and synchronization features like uploading files into the DHT, downloading them at any other peer or sharing files among multiple users. Hive2Hive only provides an application programming interface (API), but is not ready for end-users without an additional client. During this thesis, an Android client for file synchronization across multiple Android devices and computers was developed and deployed.

4.2.1 Challenges with the Dalvik Virtual Machine

Code written in Java can be compiled to be executable by the Dalvik Virtual Machine (DVM). Android applications are therefore natively developed using Java. The JVM compiler and a converter are required to make them executable in the DVM. The differences between the JVM and the DVM are particularly notable in the encryption and the serialization part of Hive2Hive.

Encryption

BouncyCastle[2] is the largest open-source cryptography API for Java. It offers additional security and features compared to the default security provider. Android comprises an optimized but cut-down implementation of BouncyCastle. For security reasons, the full implementation of the latest BouncyCastle library is required for Hive2Hive. To overcome namespace conflicts (i.e. two different BouncyCastle implementations with the same package names), SpongyCastle[3] has been released.

[1] `http://www.getsync.com/`
[2] `https://www.bouncycastle.org/`
[3] `https://rtyley.github.io/spongycastle/`

On Android devices, encryption is achieved with the SpongyCastle security provider, and *normal* computers use BouncyCastle. The encryption algorithms (like RSA or AES) are not changed, just their implementations.

Serialization

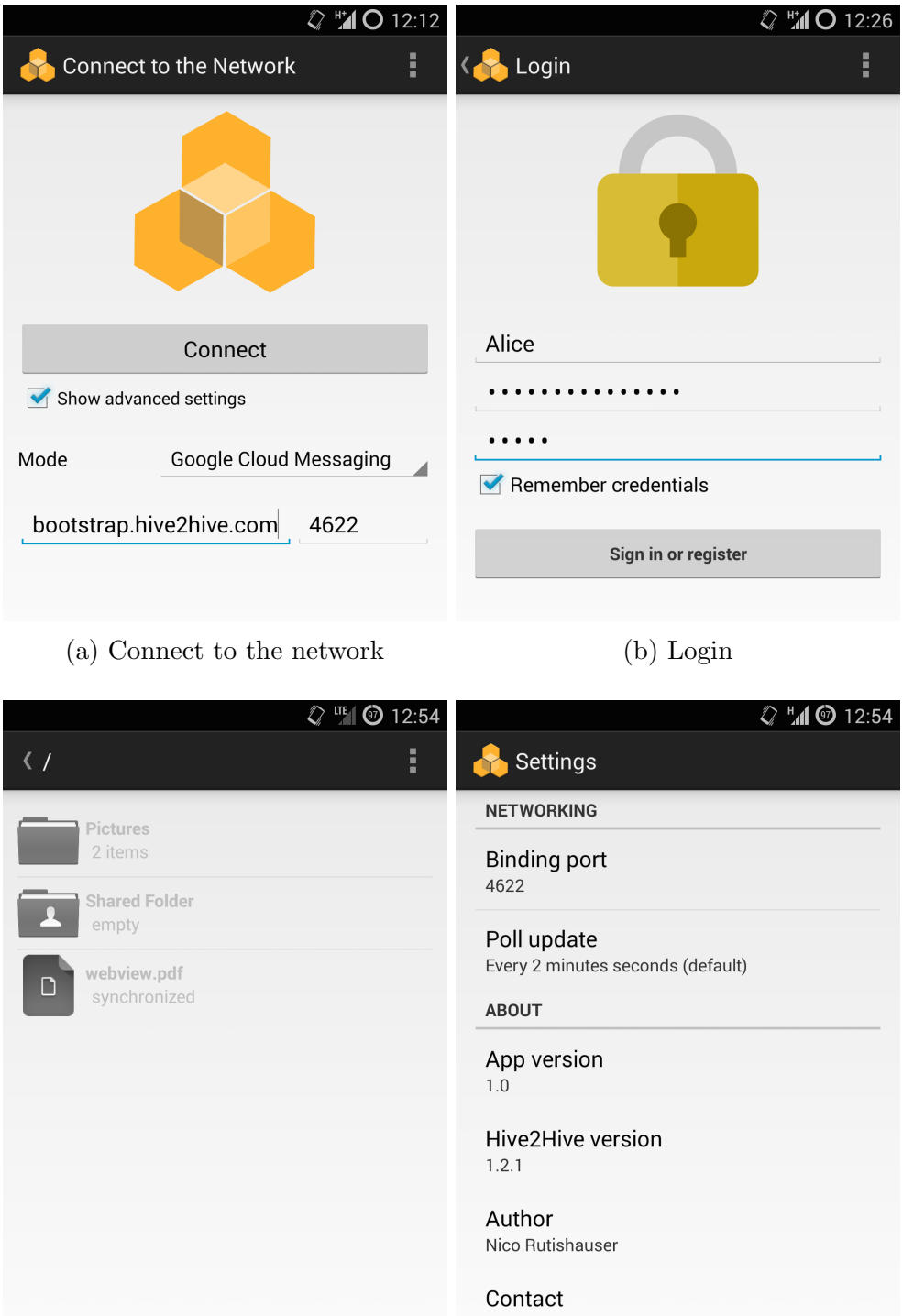
The conversion of an object to a byte stream is called *serialization*. Serialization is required whenever data needs to be persisted or sent over a network. Usually, a class identifier together with all class variable values are converted, not the class declaration itself. At the deserialization, it is expected that the same class is already available in the JVM and can be instantiated with the serialized class variables to get the exact same state as before the serialization.

The Java default serialization / deserialization identifies the class using the full package path. Having two different security providers (BouncyCastle and SpongyCastle), it is very likely that the sender and receiver peer have different security providers. If for example a key pair is sent over the network, the recipient cannot map the sender's implementation to their own implementation. This conflict is resolved by using the flexible and customizable fast-serialization library[4] (FST). Classes can be registered at the serializer and deserializer such that its own identifier (integer number) is used for the mapping. FST not only reduces the size of the serialized object, but also resolves further DVM / JVM incompatibilities. A list of the discovered incompatible classes using Java serialization can be found in appendix A.

4.2.2 Screenshots

Figure 4.1 shows four exemplary screenshots of the Android Hive2Hive client. The connection activity (Figure 4.1a) allows to select the relaying mode and to configure the bootstrap address. The routing table update interval can be configured in the settings activity (Figure 4.1d).

[4] `https://github.com/RuedigerMoeller/fast-serialization`



(a) Connect to the network

(b) Login

(c) Manage synchronized files

(d) Settings

Figure 4.1: Screenshots of the Hive2Hive Android client

Chapter 5

Evaluation

Chapter 3 introduced two challenges with peers on mobile devices: The first challenge is the NAT traversal, which can be solved using multiple relaying techniques. The second challenge is the limited power capacity of a mobile device. Using a naive NAT traversal approach (like *direct relaying* or *reverse connection relaying*), a high energy consumption at the mobile peer is expected. With message buffering and the use of the mobile operating system's push notification service, reduced battery consumption is anticipated. Details about the implementation have been provided in chapter 4. The effects of the proposed improvements compared to the native approach are shown in this chapter. Software and hardware measurements are conducted in order to evaluate the tradeoff between energy consumption and response time.

5.1 Experimental Setup

For the experiment, a small overlay network with four publicly reachable fixed-network peers and one mobile phone is used. One of these fixed-network peers acts as the relay peer to serve the unreachable mobile phone. Another fixed-network peer is the *query peer*, making *put* requests to the mobile phone. The other two peers in the network are passive seed nodes for network stability. The small network size is chosen because the number of peers in the DHT does not affect the battery consumption of the mobile phone. More peers in the overlay increases the number of control messages, but they are all handled at the relay peer anyway. For an unreachable peer, it does not matter if the DHT consists of few peers or a million peers. With few peers the responsible key range is larger, but in this experiment, the DHT activity (i.e. store data) is only generated by one query peer and always targets the mobile peer.

For the hardware measurements, a testbed consisting of a Measurement Computing USB-1608FS-Plus device and a docking station is used. The data acquisition (DAQ) device meters the current of the mobile phone at a sample rate of 10Hz and continuously transfers it over USB to a connected computer. The testbed in action can be seen in Figure 5.1. The wiring of the DAQ device, the mobile phone and it battery is shown in Figure 5.2.

Figure 5.1: A data acquisition device is wired to a docking station to measure the energy consumption of the mobile phone.

The energy measurement is started after the mobile peer has bootstrapped to the overlay network, but before the first put request is triggered. After the success of the last put request, the measurement is stopped. A description of the peer roles and their configuration options is listed in the following. Table 5.1 summarizes all parameters used in this evaluation.

Connectivity	Wifi 3G
Routing table update interval	60s 120s
Relaying technique	Reverse connecion relaying Buffered direct relaying BPN relaying
Buffer timeout	20s 40s 60s

Table 5.1: Evaluation parameters

5.1.1 Mobile Peer

The mobile phone used for the hardware measurements is a Samsung Nexus S with Android version 4.1.2. Software measurements are conducted with an Samsung Galaxy S II running Android 4.4.4 with CyanogenMod version 11. For all measurements, the display is kept on at about 30% of its maximum brightness level. Energy measurements of the

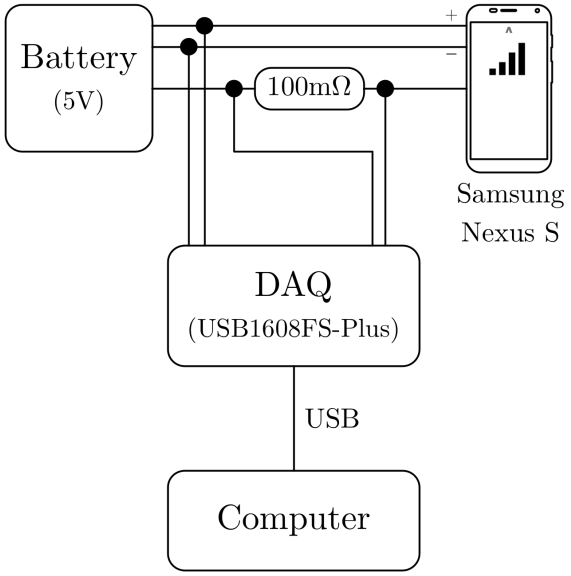


Figure 5.2: The wiring schema of the testbed. The DAQ device measures the voltage at four points between the battery ad the mobile phone.

overall battery consumption may be disturbed by other background processes. To reduce interference, background tasks have been killed, synchronization has been turned off and automatic updates have been disabled.

Connectivity

The mobile phone can either be connected to the cellular network (i.e. 3G) or to a Wifi access point. Since the power state models of these adapters differ significantly (see section 2.1), a case distinction is necessary. In Wifi, the measurements have been conducted few meters from the access point and therefore with a good signal strength. For the NAT traversal demonstration, the mobile phone is never connected to the same local area network (LAN) as the other peers. The signal when using the cellular network is less controllable due to cell breathing but was always at a high level. Cell breathing is the adaptation of the cell size to the current load [20]. If few clients are in the cell, the base station increases the sending power in order to serve more distant clients. If too many clients are in the cell, the base station reduces its sending power. The used cellular network providers were O_2 (D), Swisscom (CH) and Sunrise (CH).

Relay Mechanism

The used relay type is defined by the mobile peer during the connection setup phase. The measurements are conducted with three different relaying mechanisms. First, *reverse connection relaying* represents the unbuffered approach with a permanently open TCP

channel between the relay peer and the mobile peer. Second, *buffered direct relaying* is chosen to show the effect of buffering while still keeping the TCP channel open. The third mechanism is *BPN relaying* with GCM to show the energy improvement when omitting the open TCP channel but still applying buffering. As stated in Table 3.1, unbuffered relaying through GCM is not possible because throttling would be applied after few consecutive push notification messages.

Routing Table Update Interval

The routing table update interval should not disturb the measured energy consumption effects of other parameters because its function is not dependent on the relaying mechanism but an unavoidable necessity for the DHT stability. A short interval would engender a polling strategy because the buffer is emptied before the notification message can be sent. The chosen intervals are either 60s or 120s.

5.1.2 Relay Peer

For unbuffered approaches, the relay peer has no other option than just forwarding requests. If the approach involves a buffer, the relay peer manages its size and timeout. In the measurements, only the timeout parameter is varied as it is easier to control and does not depend on the DHT activity. The buffer size is set to infinite. The energy consumption of the mobile device with buffer timeouts of 20s, 40s and 60s are measured.

5.1.3 Query Peer

The query peer is responsible to generate load at the relay and the mobile peer. With a regular interval, data is stored onto the mobile peer using the *put* interface of the DHT. Put requests are chosen for two reasons: first, it can only be answered by the peer responsible for the location key. Get requests could theoretically be answered by replicas or through caching. The second reason is to avoid running into a bottleneck at the network interface of the mobile peer. The downlink data transfer rate is generally higher than the uplink rate. The data size of the object to store on the mobile peer is kept at the constant value of 128 bytes in all measurements.

Put Interval

The put interval is the pace the query peer stores data on the mobile peer. The location key of the data to store is the mobile peer's logical location in the overlay. The other peers are configured to deny any put request and only the mobile peer is ready to accept them. To simulate a high DHT activity, an interval of 2 seconds is used.

Number of Puts

Having a constant put interval, the number of puts defines the total time of one measurement run. On one hand, the number of puts should be chosen large enough to reduce the effect of uncontrollable background processes of the Android system. On the other hand, the number should be small enough such that all measurement runs are feasible. Therefore, the number of puts is set to a constant value of 100. With a put interval of 2 seconds, one measurement run has a theoretical total runtime of 198 seconds. Buffering, network delays and processing time increase the total time to respond all requests.

5.2 Measurement Error

The measurements have been conducted on multiple dates, various daytimes and at three different locations. While the distance of the mobile phone to the Wifi access point was kept approximately equidistant, the distances to the cellular network antennas is unknown. The mobile phone needs to increase the sending power if the antenna is further away, requiring more energy. Thus, the energy measurements are only comparable among other measurements at the same location. Nonetheless, multiple runs with the exact same settings have been performed at the same location to reduce the influence of possible confounders like background tasks, cell breathing, changing weather conditions or measurement errors.

The maximum measurement error of the used hardware measurement setup can be quantified. Kaup et al. [13] used the exact same setup for measuring the energy consumption of a Raspberry Pi. The maximum absolute measurement error adds up to 2.47% of the measured energy consumption.

5.3 Results

In total, more than 120 measurements have been conducted, leading to a macroscopic overview of how much energy the proposed relay mechanisms consume compared among each other. The idle energy consumption results (section 5.3.1) show that a separation of Wifi and 3G is important because they have two very different baseline energy consumptions. After showing the energy consumption of a mobile phone connected to an idle DHT, all results under load with Wifi connectivity are presented. Then, the measurement results of 3G are revealed. All charts presented in this section are hardware measurements. Due to their inaccuracy, software measurements can be found in appendix B.3.

5.3.1 Idle Energy Consumption

Figure 5.3 shows the energy consumption of a mobile phone connected to the DHT. In these four measurements, no requests are fired by the query peer and the mobile peer only

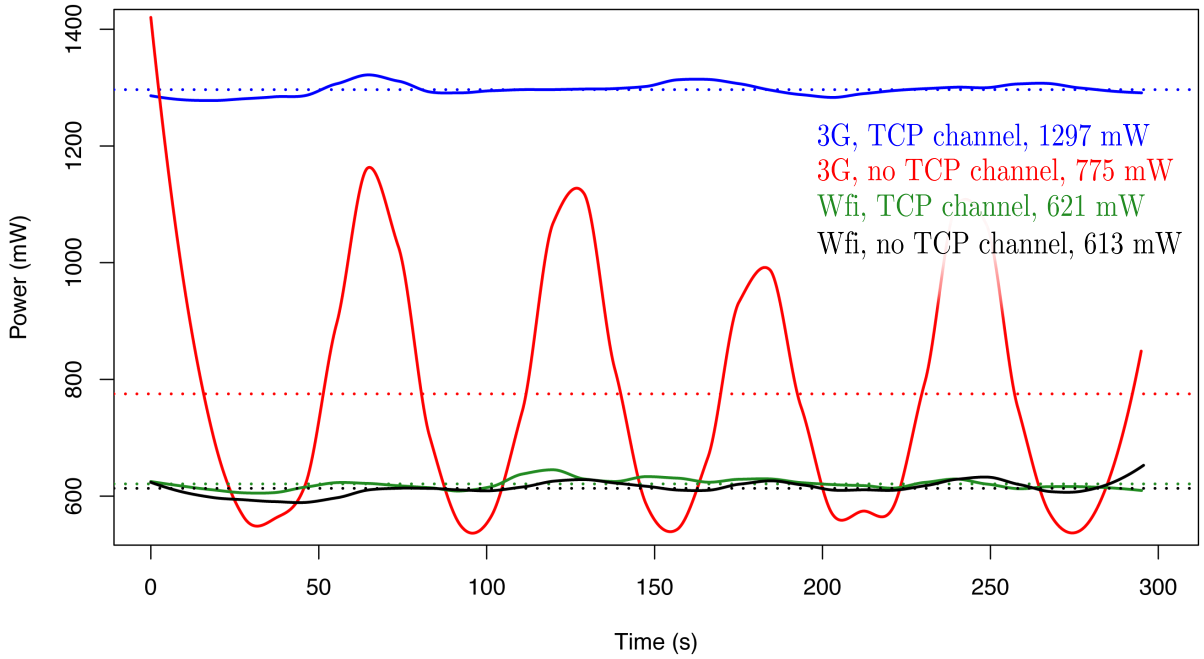


Figure 5.3: Interpolated measurements of a mobile peer connected to the DHT but not performing any other work than sending the routing table to the relay peer each 60s. The dotted lines are the averages over the total measurement time of 5 minutes.

performs the routing table update every 60s. The four lines are the combinations of Wifi / 3G and maintaining an open TCP connection or not. No TCP connection is required in the BPN relaying approach. While the open connection does not consume extra energy in Wifi, the energy levels in 3G differ heavily. Maintaining the open connection in the cellular network continuously consumes almost 1300 mW (blue line). The oscillating red line shows that although the mobile peer needs to regularly connect to the relay peer to send the newest routing table, the average energy consumption is only 775 mW. The large difference in 3G cannot be reproduced in Wifi, where the slightly higher average with an open TCP channel is within the measurement error tolerance.

While Figure 5.3 shows the result of one measurement run, average numbers of three measurement runs with Wifi and 3G are presented in Tables 5.2 and 5.3, respectively. The exact numbers for idle energy consumption with a routing table update interval of 120s can be found in Tables B.1 and B.2.

	Run 1	Run 2	Run 3	Average
TCP channel	687 mW	621 mW	645 mW	651±33 mW
No TCP channel	652 mW	613 mW	610 mW	625±24 mW
Savings	5.05%	1.23%	5.50%	3.93%

Table 5.2: The average idle energy consumption of three measurement runs in Wifi with a routing table update interval of 60s. Each measurement run was conducted at a different access point. The last row shows the savings of having no open TCP channel. The average savings of omitting the open TCP channel in Wifi is only 3.93%.

	O_2	Sunrise	Swisscom	Average
TCP channel	988 mW	1297 mW	1063 mW	1116±161 mW
No TCP channel	762 mW	775 mW	703 mW	747±39 mW
Savings	22.84%	40.22%	33.87%	32.31%

Table 5.3: The average energy consumption of a connected mobile peer in 3G measured with three different cellular network providers. The routing table update interval is 60s. In average, more than 32% of the energy is saved when not having an open TCP channel in a 3G network.

5.3.2 Results in Wifi

A comparison of the energy consumption of the proposed relaying mechanisms in Wifi can be found in Figure 5.4. Each plot shows the measured energy curve of the mobile phone from the first put request to the last put response. In addition, the average and total power consumption and the total duration to respond all requests are shown. For clarity and fast comparison, a horizontal line shows the average power consumption. Each data point in the curve is the mean value of ten measured data points. With the sample rate of 10Hz, this results in one plotted point per second.

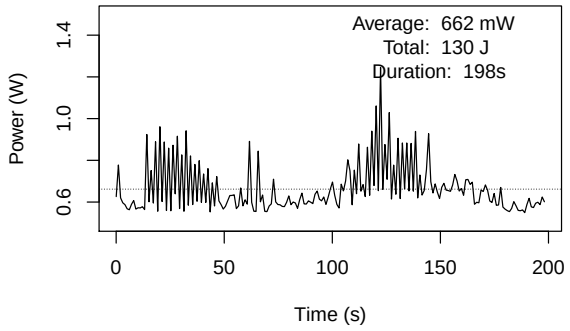
As a reference, a full peer running on a mobile phone is also measured, where NAT traversal has been bypassed by manually setting up port forwarding at the NAT device. Running a full peer not feasible in real-world applications because NAT traversal must be achieved without manual configuration. Figure 5.4a shows that in full-peer mode (without any dependence on the on the relay peer), the mobile phone has an average energy consumption of 662 mW.

The measurement in Figure 5.4b has the highest energy consumption because reverse connection relaying requires an open TCP channel to the relay peer. As the requests are forwarded to the mobile peer immediately, the total duration to answer the 100 put requests takes exactly as long as in the full peer mode.

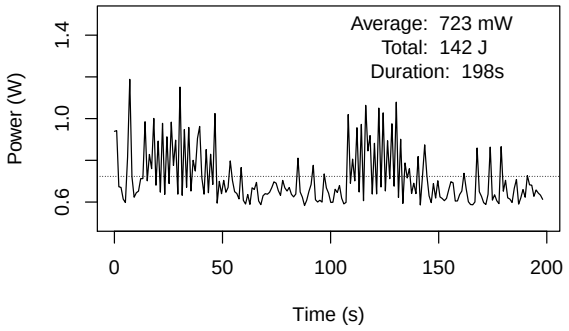
The measurements in Figures 5.4c to 5.4h show the energy consumption with a buffer at the relay peer. The longer the buffer timeout, the lower the average energy consumption. But since the time to complete all requests increases with the buffer timeout, the total energy consumption between the first and the last put grows again. Interestingly, the total and average energy consumption both exhibit minima when the buffer size is set to 40s.

Spikes indicate temporary high energy consumptions, as for example it would be the case when the buffer is downloaded from the relay peer and the replies are sent back to the query peer. The different buffer timeouts in the charts 5.4c to 5.4h can be seen by regarding the regular distance between the spikes, but an indicator for the slightly differentiating energy consumption is unapparent.

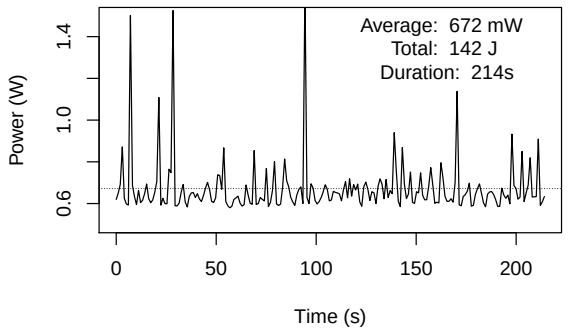
Table 5.4 shows the average results of three measurement runs (see Tables B.3 to B.5). The columns $\text{Savings}_{\text{avg}}$ and $\text{Savings}_{\text{total}}$ indicate how the relaying mechanism performs compared to reverse connection relaying. The highest energy savings can be observed



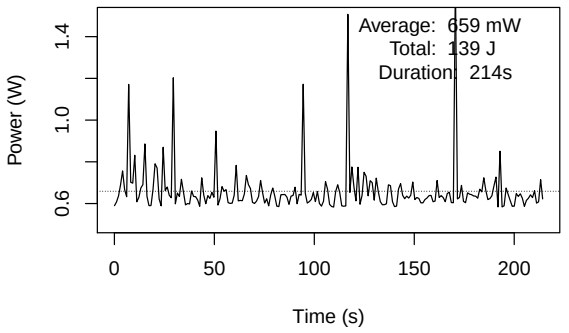
(a) Full peer (no relaying)



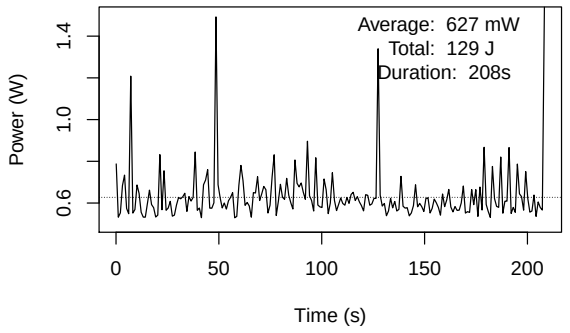
(b) Reverse connection relaying (non-buffered)



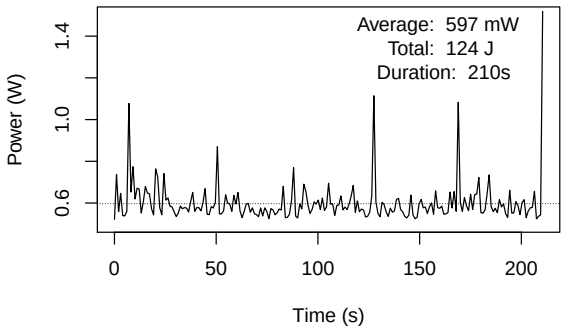
(c) Buffered direct relaying (20s buffer)



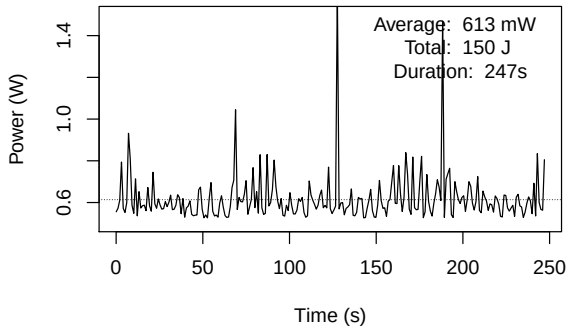
(d) BPN relaying (20s buffer)



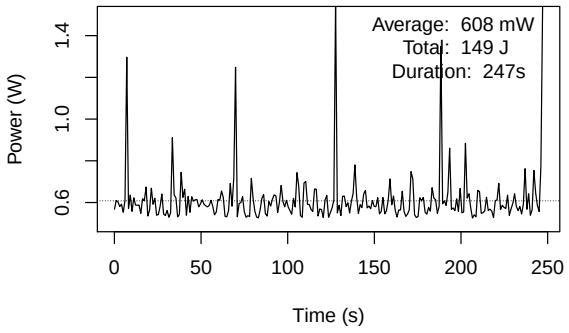
(e) Buffered direct relaying (40s buffer)



(f) BPN relaying (40s buffer)



(g) Buffered direct relaying (60s buffer)



(h) BPN relaying (60s buffer)

Figure 5.4: Comparison of the energy consumption of three relaying mechanisms (reverse connection relaying, buffered direct relaying and BPN relaying) with a non-relayed full peer. The latter measurement has been conducted using port forwarding at the Wifi router. The routing table update interval for all measurements is 120s.

when using BPN relaying with a buffer timeout of 40s. Buffered direct relaying and BPN relaying with a buffer timeout of 60s both consumed 13% and 11% more energy in total, although operating at a lower average energy consumption. The reason is the higher total time required to answer all requests because of longer buffering time.

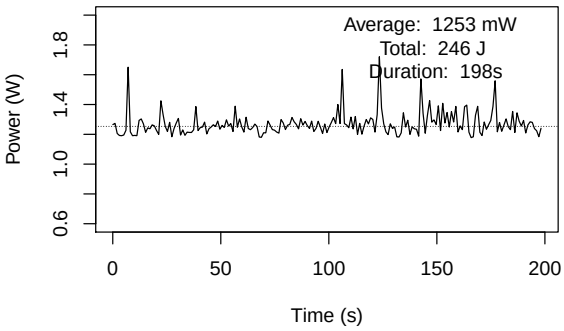
	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$	$\text{Savings}_{\text{avg}}$	$\text{Savings}_{\text{total}}$
Full peer (no relaying)	662 mW	198 s	130 J	4.64%	4.91%
Reverse connection r.	694±33 mW	198 s	136±6 J	0%	0%
Buffered direct r. (20s)	656±15 mW	213 s	138±4 J	5.54%	-1.07%
Buffered direct r. (40s)	631±10 mW	209 s	130±2 J	9.12%	4.55%
Buffered direct r. (60s)	633±17 mW	247 s	155±4 J	8.78%	-13.38%
BPN relaying (20s)	647±25 mW	213 s	136±5 J	6.88%	0.29%
BPN relaying (40s)	620±37 mW	209 s	128±6 J	10.75%	6.02%
BPN relaying (60s)	624±29 mW	245 s	152±5 J	10.05%	-10.98%

Table 5.4: Mean values of three measurement runs in Wifi shown in Tables B.3, B.4 and B.5. The last two columns show the power savings of the average and total power consumption compared to reverse connection relaying. Positive values stand for energy conservation, negative values signify that more energy than reverse connection relaying has been required.

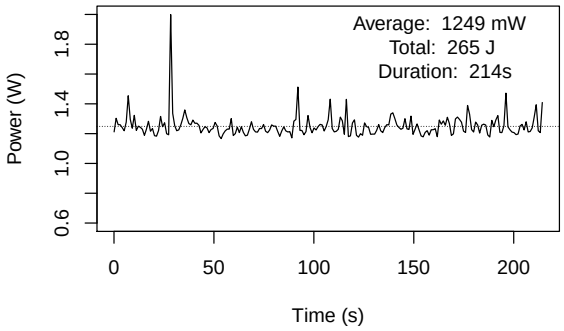
5.3.3 Results in 3G

A higher variance of the energy consumption among the relaying mechanisms compared to Wifi is measured with 3G network connectivity. Similar to the previous section, Figure 5.5 shows an overview of one measurement run. The measurement conditions are the same, except that the mobile phone is connected to the cellular network provider instead of a Wifi access point. Figures 5.5a, 5.5b, 5.5d and 5.5f indicate that keeping an open TCP connection to the relay peer is only possible with heavy battery drain. Even with buffering, where the connection is solely used in bursts, the high energy consumption continues. In comparison, the use of the mobile operating's system optimized push notification channel (BPN relaying) only drains battery if a pool of requests arrive. The peaks in the BPN relaying approach are higher than than in buffered direct relaying, but the average energy consumption is at a much lower level. The distances between the peaks depend on the configured buffer timeout of 20s, 40s or 60s. In the inactive state, the energy consumption is similar as during inactive phases in the idle measurements in Figure 5.3.

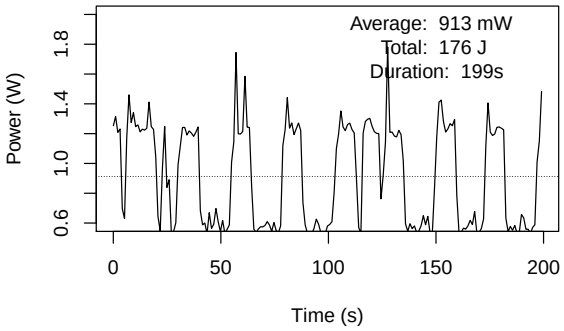
The measurement results of one energy measurement run are shown in Table 5.5. Compared to the reverse connection relaying approach, the buffered direct relaying approach performs poorly. Although the average energy consumption can slightly be reduced, the total power consumption increases by up to 23%. Much more energy can be conserved in 3G networks when using the BPN relaying approach. The lowest average energy consumption with 808 mW has BPN relaying with a buffer of 60s, but the minimum total energy consumption to answer all requests has been measured at the BPN relaying approach with a buffer timeout of 20s. The table shows that up to 28% of the total energy consumption can be saved when using push notifications instead of reverse connection relaying.



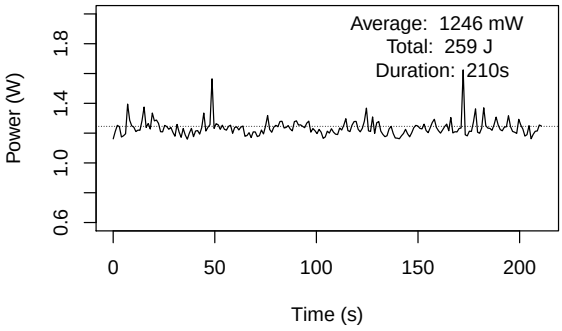
(a) Reverse connection relaying (non-buffered)



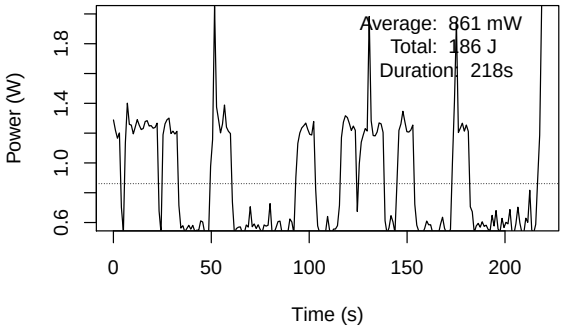
(b) Buffered direct relaying (20s buffer)



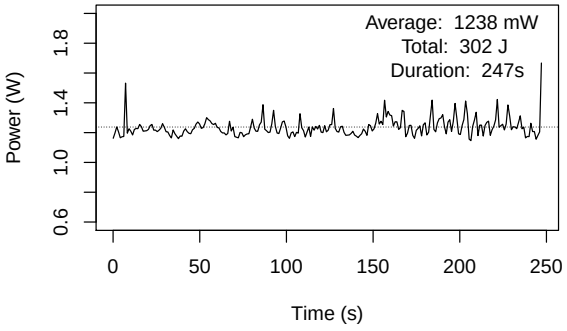
(c) BPN relaying (20s buffer)



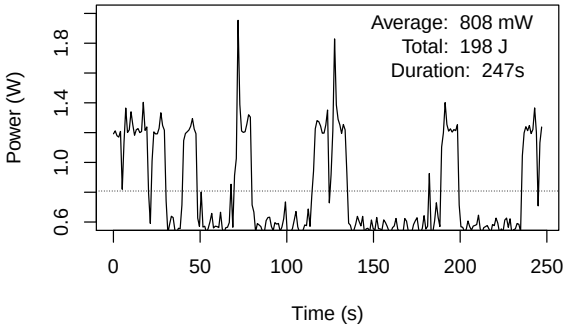
(d) Buffered direct relaying (40s buffer)



(e) BPN relaying (40s buffer)



(f) Buffered direct relaying (60s buffer)



(g) BPN relaying (60s buffer)

Figure 5.5: Energy consumption of three relaying mechanisms in comparison using a 3G network. While buffering alone does not reduce the energy consumption, a clear improvement can be seen with the BPN relaying approach.

More measurements which compare reverse connection relaying to BPN relaying with other network providers can be found in Tables B.6, B.7 and B.8. They confirm the findings that the BPN approach saves a considerable amount of energy.

A full peer as a reference is not possible in 3G networks because the NAT and firewall of the mobile network operator cannot be controlled.

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$	$\text{Savings}_{\text{avg}}$	$\text{Savings}_{\text{total}}$
Reverse connection relaying	1253 mW	198 s	246 J	0%	0%
Buffered direct relaying (20s)	1249 mW	214 s	265 J	0.36%	-7.66%
Buffered direct relaying (40s)	1246 mW	210 s	259 J	0.62%	-5.46%
Buffered direct relaying (60s)	1238 mW	247 s	302 J	1.26%	-22.73%
BPN relaying (20s)	913 mW	199 s	176 J	27.18%	28.59%
BPN relaying (40s)	861 mW	218 s	186 J	31.33%	24.23%
BPN relaying (60s)	808 mW	247 s	198 J	35.52%	19.65%

Table 5.5: A comparison of the energy consumption of the proposed relaying techniques in 3G. The last two columns show the savings compared to reverse connection relaying, which is taken as the baseline.

5.4 Discussion

The usage of a buffer at the relay peer is counterproductive in Wifi environments. The buffer slightly reduces the average energy consumption, but increases the total time to answer all requests such that the mobile device needs to be active for a longer timespan. The maximum energy savings is only 6% and not significant when taking the measurement error and distortion factors into consideration. The small difference between reverse connection relaying and BPN relaying can be explained with the Wifi state model (see Figure 2.1c). The state is demoted to LOW within some milliseconds of inactivity. Maintaining a steady TCP connection to the relay peer is cheap in Wifi because the wireless network adapter only needs to be active for a few milliseconds when the heartbeat is sent.

When using 3G, the energy consumption can be greatly reduced if a steady TCP connection is relinquished. A reduction of up to 30% has been measured with BPN relaying. The idle energy measurements already indicate a higher baseline energy consumption when maintaining a TCP channel. The extent of its usage seems to have no impact on the total energy consumption. Therefore, the buffered direct relaying approach is adverse. The BPN measurements reveal that the buffer timeout of 40s performed best. Sometimes, the total runtime with a buffer of 40s was even shorter than with 20s, which can be explained with failing GCM messages when sending them at a fast pace of 20s. Although initial tests indicated that a GCM sending rate of 20s seemed to be reliable, this shows again the unpredictability of using a third party service.

The unreliability of GCM is a large problem for applications that depend on it. Established messengers such as WhatsApp[1] or Facebook Messenger[2] also renounced using GCM and

[1] `http://www.whatsapp.com/`
[2] `https://www.facebook.com/`

decided to use their own (highly optimized) push notification services. Their impact on the battery lifetime is not known but is kept within an acceptable limit, otherwise the number of user complaints would be remarkably higher. This shows that technically it is possible to have a dedicated TCP connection without the high energy consumption encountered in this thesis. The permanently open TCP connection used in this thesis was not optimized for mobile devices.

A tradeoff has to be made between the average energy consumption and the response latency. If buffering is applied, a minimum latency of 20s must be acceptable. The highest accepted latency to respond a request depends on the use case and on how the application interacts with the DHT. A transaction that includes multiple successive DHT calls suffers from a high total response time as soon as a buffer is in play. In the worst case, every request ends up in a separate buffer iteration. If the transaction logic allows parallel requests, ideally they are all buffered together. In that case, the total transaction time is at most one buffer timeout. For example, Hive2Hive does many double-checks in order to increase the reliability of the P2P network. If a content is removed, another *get* call is made to check if the data is really deleted and nothing went wrong. Further, Hive2Hive uses the versioned DHT approach [6] where every put needs to be confirmed if no version conflict at any replica peer has been detected. Every request can therefore take up to the double amount of time.

Replication can reduce the latency of *get* requests dramatically. With replication, neighbor peers keep a copy of the content, such that nothing is lost if the responsible peer goes offline. The newly closest peer is then responsible to again replicate the data to another neighbor in order to satisfy the replication factor. If data is not only stored at the unreachable mobile peer, but also at several fixed-network replicas, it can be obtained from there without the need for buffering. In addition, this helps to reduce the battery consumption of the mobile device because requests are offloaded to its replica peers.

All presented measurements have been conducted with a regular DHT activity of one put every two seconds. In a real-world scenario, the request interval would not be constant. Normally, the location keys for storing data are assigned randomly, thus the responsible peer is randomly chosen from all peers in the DHT. In consideration of the energy consumption, regular requests are the worst-case scenario. If requests arrive at random timings, the probability that more requests can be collected in one buffer is higher. The average time where the buffer remains empty is also higher. Therefore, the total activity at the mobile peer decreases as requests arrive more randomly. In addition to the regularity, the experiment has been designed with a rather high DHT activity. A lower DHT activity even increases the energy consumption gap between relaying techniques with an own TCP connection and relaying techniques using the shared push notification service. Once the dedicated TCP channel is set up, costs for maintaining it in 3G are very high and hardly depend on its usage. Maintaining this channel if it is only used for a few requests each minute or hour increases the energy consumption per request ratio dramatically. With BPN relaying, such maintenance costs do not exist. The energy consumption per request ratio only depends on the number of requests in the buffer. The ratio has an upper boundary (i.e. one request in the buffer) and improves if more requests are buffered at once. This experiment therefore showed the *minimum* possible energy savings when using the BPN relaying approach. In practice, as for example with the Hive2Hive Android

client, the energy savings are expected to be higher.

For 3G networks, a clear recommendation to prefer BPN relaying over reverse connection relaying can be given. In Wifi, the additional latency due to buffering is not worth the slight energy savings. It is easiest to rely on reverse connection relaying (or direct relaying) or act as a full peer with other NAT traversal techniques like hole punching or UPnP.

Unfortunately, measurements with LTE connectivity were not possible because the testbed is not built or configured for an LTE capable device. Despite this limitation, assumptions can be made. The cost for maintaining a continuous TCP connection are expected to be less with LTE than with 3G because the sleep-cycles in the DRX states allow a temporary energy reduction if no traffic is required (see Figure 2.1b). The peak energy consumption of an LTE cellular adapter is significantly higher than with 3G, but with buffering, the number of peaks might be reduced and therefore also the total energy consumption. If these two assumptions are fulfilled, buffered direct relaying and BPN relaying are expected to have a similar energy chart.

Chapter 6

Summary, Conclusions and Future Work

6.1 Summary

This thesis proposed several relaying techniques for NAT traversal. The focus remained on finding an energy-efficient relaying technique such that it is feasible to run a peer on a mobile phone. With the introduction of a variable buffer at the relay, multiple DHT requests can be collected and transmitted to the mobile peer at once. To further reduce the traffic at the mobile peer, the relay peer undertakes DHT maintenance tasks like answering aliveness checks or routing requests.

The mobile peer opens a dedicated long-living TCP connection to the relay peer, such that the latter can push requests to the mobile phone. Most mobile phone operating systems offer a built-in and highly energy-optimized push notification service. In the proposed BPN relaying, the dedicated TCP connection is replaced by Google Cloud Messaging to push notifications to Android-based mobile phones.

Multiple energy measurements have been conducted that allow the comparison of the relaying techniques in a constant DHT scenario. A significant energy difference between techniques that use an own TCP channel and techniques relying on the built-in push notification service have been found. If the mobile phone is connected to the Internet over 3G, the energy consumption could be reduced by up to 30% when using the push notification service. In Wifi networks, the energy savings are at most 6%, which is not significant.

6.2 Conclusions

Although battery technology is expected to make further progress in the future, the energy limitation of mobile phones will remain. Smartphone application developers always need to consider the energy-efficiency, especially if networking resources are used. BPN relaying

not only solves the NAT traversal problem for mobile peers participating in a P2P network, but also reduces the total energy consumption by 30% in 3G with a high DHT activity. A low DHT activity and irregular request arrival timings further reduce the average energy consumption.

With the implementation of a distributed file synchronization client for Android devices, it has been shown that the BPN approach not only works in theory, but can be applied in the real-world.

6.3 Future Work

The relaying techniques may be further improved with the addition of intelligence at the relay peer. To enhance failure safety, each unreachable peer can be connected to multiple relay peers concurrently. Presently, if a buffered relaying technique is applied, each relay peer maintains its own buffer and notifies the unreachable peer if the buffer times out or is full. In the worst case, the unreachable peer is constantly busy obtaining and answering all these buffers. In a future work, the relay peers could have one *shared buffer* where all requests are collected. The unreachable peer would be notified only once to obtain this shared buffer. With this approach, the relay peers face a larger workload due to the additional buffer and notification synchronization task, but the load at the unreachable peer would be reduced.

In a future work, more parameters of the experiment could be varied. The impact of the data size on the energy consumption has not been investigated in this thesis. Further on, the put interval and its regularity could be varied to better simulate real applications. There are many tradeoffs between the possible experiment parameters that need to be investigated, as for example the determination of the buffer timeout at a minimum energy consumption having random request timings. Another tradeoff exists between the reliability when having multiple relay peers and the energy consumption of the additional maintenance costs. Moreover, how does the churn rate influence the energy consumption at the mobile peer?

Due to hardware limitations, it was not possible to conduct the experiment with LTE connectivity. Its state model is a mixture between the Wifi and the 3G state model, which would lead to an interesting comparison under the same experiment conditions. LTE is expected to have a reduced energy consumption compared to 3G because of the sleep cycles during short inactive phases. The energy consumption would probably not undercut Wifi because LTE needs a higher signaling power in order to be able to communicate with the base station.

Bibliography

[1] B Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60. IEEE, 2003.

[2] Thomas Bocek. TomP2P - A P2P-based high performance key-value pair storage library.

[3] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1548–1557. IEEE, 2001.

[4] David Ferguson. P2p file sharing–the evolving distribution chain. *CacheLogic, WDC*, 2006.

[5] Alexandre Gerber, Subhabrata Sen, and Oliver Spatscheck. A call for more energy-efficient apps, April 2011.

[6] Sebastian Golaszewski. Consistency in distributed systems.

[7] Sebastian Golaszewski, Christian Lüthold, and Nico Rutishauser. Hive2Hive - an open-source library for distributed file synchronization and sharing. UZH, 2014.

[8] Christian Gross, Fabian Kaup, Dominik Stingl, Bjorn Richerzhagen, David Hausheer, and Ralf Steinmetz. EnerSim: An energy consumption model for large-scale overlay simulators. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 252–255. IEEE, 2013.

[9] Zhou Hu. NAT traversal techniques and peer-to-peer applications. *Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology*, 2005.

[10] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 225–238. ACM, 2012.

[11] Cisco Visual Networking Index. Forecast and methodology, 2013-2018, 2014.

[12] Otso Kassinen, Erkki Harjula, Jari Korhonen, and Mika Ylianttila. Battery life of mobile peers with UMTS and WLAN in a Kademlia-based P2P overlay. In *Personal, Indoor and Mobile Radio Communications, 2009 IEEE 20th International Symposium on*, pages 662–665. IEEE, 2009.

[13] Fabian Kaup, Philip Gottschling, and David Hausheer. PowerPi: Measuring and modeling the power consumption of the Raspberry Pi. In *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pages 236–243. IEEE, 2014.

[14] Imre Kelényi and Jukka K Nurminen. Energy aspects of peer cooperation measurements with a mobile DHT system. In *Communications Workshops, 2008. ICC Workshops' 08. IEEE International Conference on*, pages 164–168. IEEE, 2008.

[15] Imre Kelényi and Jukka K Nurminen. Optimizing energy consumption of mobile nodes in heterogeneous Kademlia-based distributed hash tables. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST'08. The Second International Conference on*, pages 70–75. IEEE, 2008.

[16] Ding Li and William GJ Halfond. An investigation into energy-saving programming practices for Android smartphone app development.

[17] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[18] Jukka K Nurminen and Janne Noyranen. Energy-consumption in mobile peer-to-peer-quantitative results from file sharing. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 729–733. IEEE, 2008.

[19] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Characterizing radio resource allocation for 3G networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 137–150. ACM, 2010.

[20] Jochen H Schiller. *Mobile communications*. Pearson Education, 2003.

[21] Andrew Tanenbaum and Maarten Van Steen. *Distributed systems*. Pearson Prentice Hall, 2007.

[22] Arno Wacker, Gregor Schiele, Sebastian Holzapfel, and Torben Weis. A NAT Traversal Mechanism for Peer-To-Peer Networks. In *Peer-to-Peer Computing*, pages 81–83, 2008.

[23] Matthias Wichtlhuber, J Ruckert, Dominik Stingl, Matthias Schulz, and David Hausheer. Energy-efficient mobile P2P video streaming. In *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, pages 63–64. IEEE, 2012.

[24] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei 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*, pages 105–114. ACM, 2010.

Abbreviations

ALG	Application Level Gateway
API	Application Programming Interface
BPN	Buffered Push Notification
CAGR	Compound Annual Growth Rate
DAQ	Data Acquisition
DHT	Distributed Hash Table
DR	Direct Relaying
DVM	Dalvik Virtual Machine
EIRP	Equivalent Isotropically Radiated Power
FST	Fast Java serialization drop-in replacement
GCM	Google Cloud Messaging
LAN	Local Area Network
LTE	Long Term Evolution
NAT	Network Address Translation
RCR	Reverse Connection Relaying
UPnP	Universal Plug and Play
Wifi	Wireless Fidelity, Wireless Local Area Network

Glossary

Cell breathing Cell breathing is the adaptation of the cell size to the current load. If few clients are in the cell, the base station increases the sending power in order to serve more distant clients. If too many clients are in the cell, the base station reduces its sending power.

Distributed Hash Table A Distributed Hash Table is a P2P overlay structure serving as a key-/value storage. The key space has an upper limit and each peer is responsible for a certain key range. Lookups can be performed by asking the responsible peer for the stored value.

Hint message A hint message is technically the same as a *tickle message*. In this thesis, the term *hint message* is used in reverse connection relaying to notify the unreachable peer about a request.

Leaky bucket algorithm The leaky bucket algorithm serves as a throttling mechanism in order to limit the bandwidth and burstiness of the traffic. Messages can be sent with a constant rate. Messages that arrive at a faster pace or in bursts are buffered in the bucket. If the bucket is full, overflowing messages are simply dropped.

Mobile peer A mobile peer is a peer running on a mobile phone. Mobile peers are usually behind a NAT and require a relay peer to actively participate in the P2P network.

Network Address Translation Since the number of IPv4 addresses is limited, not every computer can have a unique IP address. Therefore, the total IP address range is split into a public and a (reusable) private part. NAT devices (usually routers) have a public address, devices behind the NAT have a private address. The NAT device translates from the private to the public IP address space and vice versa.

Relay peer A relay peer is publicly reachable and helps an unreachable peer to participate in the P2P network by acting as a proxy.

Routing table update The process of sending the routing table of the unreachable peer to its relay peer, such that the relay peer can answer routing requests on behalf of the unreachable peer.

Super-peer A well-reachable peer that has a special role (e.g. for serving unreachable peers) or additional server-like functionality.

Tickle message A tickle message does not contain any payload but has the purpose to wake up the receiver to perform a pre-defined work. In this thesis, a *tickle message* is used in Google Cloud Messaging for notifying the mobile peer to obtain the buffer from the relay peer.

List of Figures

List of Tables

Appendix A

DVM - JVM Serialization Incompatibilities

The following is a list of classes which cannot be serialized at a DVM and deserialized at the JVM (or vice versa) using the default Java serialization. The list may be incomplete, because an exception is only raised at an unsuccessful deserialization attempt. Testing all classes would exceed the scope of this work. The errors have been raised with Java version 1.7 and Android version 4.4.4.

BigInteger A BigInteger can hold numbers that exceed the range of Integer (32bit) and Long (64bit) values. The representation of a BigInteger in the JVM serializes its class variables using a key-value storage pattern whereas the Android counterpart does not. At the deserialization, the JVM expects to read this key-value storage which is non-existent if the byte stream comes from a DVM.

BitSet A BitSet is used for maintaining an array of bits. The advantage over an array of booleans is that the size of the BitSet automatically grows if required. Similar to the BigInteger, the DVM implementation relinquishes a key-value storage but writes the bit array directly into the byte stream. A deserialization at the JVM crashes because the key-value pattern is invalid.

InetAddress The implementation of the InetAddress class on Android mainly differs in the IPv6 part. Although the use of IPv6 can be restricted in the DVM and JVM, some classes still need to access IPv6 fields. Other than with the BigInteger and the BitSet, deserializing a crippled InetAddress object in the JVM does not throw an exception. An exception is only thrown as soon as the address is used to connect to a socket. Since the exception happens deep in the JVM, it is considered as a fatal exception causing the JVM to stop. An exemplary error message can be seen in Listing A.1.

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  SIGSEGV (0xb) at pc=0x000000010b305069, pid=16678, tid=22275
#
# JRE version: Java(TM) SE Runtime Environment (7.0_71-b14) (build 1.7.0_71-b14)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (24.71-b01 mixed mode bsd-amd64 compressed oops)
# Problematic frame:
# V  [libjvm.dylib+0x305069]  jni_GetObjectField+0x80
#
# Failed to write core dump. Core dumps have been disabled. To enable core dumping, try "ulimit -c
    unlimited" before starting Java again
#
# If you would like to submit a bug report, please visit:
#   http://bugreport.sun.com/bugreport/crash.jsp
#

---------------  T H R E A D  ---------------

Current thread (0x00007fe80a851000):  JavaThread "NETTY-TOMP2P - worker-client/server - -1-1" [
    _thread_in_vm, id=22275, stack(0x000000011556d000,0x000000011566d000)]

(registers skipped)

Stack: [0x000000011556d000,0x000000011566d000],  sp=0x000000011566bb90,  free space=1018k
Native frames: (J=compiled Java code, j=interpreted, Vv=VM code, C=native code)
V  [libjvm.dylib+0x305069]  jni_GetObjectField+0x80
C  [libnet.dylib+0x49dd]  getInet6Address_ipaddress+0x55
C  [libnet.dylib+0x580e]  NET_InetAddressToSockaddr+0xaf
C  [libnio.dylib+0x2cc5]  Java_sun_nio_ch_Net_connect0+0x51
j  sun.nio.ch.Net.connect0(ZLjava/io/FileDescriptor;Ljava/net/InetAddress;I)I+0
j  sun.nio.ch.Net.connect(Ljava/net/ProtocolFamily;Ljava/io/FileDescriptor;Ljava/net/InetAddress;I)I+25
j  sun.nio.ch.Net.connect(Ljava/io/FileDescriptor;Ljava/net/InetAddress;I)I+6
j  sun.nio.ch.SocketChannelImpl.connect(Ljava/net/SocketAddress;)Z+225
j  io.netty.channel.socket.nio.NioSocketChannel.doConnect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;)Z+22
j  io.netty.channel.nio.AbstractNioChannel$AbstractNioUnsafe.connect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)V+53
j  io.netty.channel.DefaultChannelPipeline$HeadContext.connect(Lio/netty/channel/ChannelHandlerContext;
    Ljava/net/SocketAddress;Ljava/net/SocketAddress;Lio/netty/channel/ChannelPromise;)V+8
j  io.netty.channel.AbstractChannelHandlerContext.invokeConnect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)V+11
j  io.netty.channel.AbstractChannelHandlerContext.connect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)Lio/netty/channel/ChannelFuture;+53
j  io.netty.channel.ChannelDuplexHandler.connect(Lio/netty/channel/ChannelHandlerContext;Ljava/net/
    SocketAddress;Ljava/net/SocketAddress;Lio/netty/channel/ChannelPromise;)V+5
j  io.netty.channel.AbstractChannelHandlerContext.invokeConnect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)V+11
j  io.netty.channel.AbstractChannelHandlerContext.connect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)Lio/netty/channel/ChannelFuture;+53
j  io.netty.channel.ChannelDuplexHandler.connect(Lio/netty/channel/ChannelHandlerContext;Ljava/net/
    SocketAddress;Ljava/net/SocketAddress;Lio/netty/channel/ChannelPromise;)V+5
j  io.netty.channel.AbstractChannelHandlerContext.invokeConnect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)V+11
j  io.netty.channel.AbstractChannelHandlerContext.connect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)Lio/netty/channel/ChannelFuture;+53
j  io.netty.channel.ChannelOutboundHandlerAdapter.connect(Lio/netty/channel/ChannelHandlerContext;Ljava
    /net/SocketAddress;Ljava/net/SocketAddress;Lio/netty/channel/ChannelPromise;)V+5
j  io.netty.channel.AbstractChannelHandlerContext.invokeConnect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)V+11
j  io.netty.channel.AbstractChannelHandlerContext.connect(Ljava/net/SocketAddress;Ljava/net/
    SocketAddress;Lio/netty/channel/ChannelPromise;)Lio/netty/channel/ChannelFuture;+53
j  io.netty.channel.DefaultChannelPipeline.connect(Ljava/net/SocketAddress;Ljava/net/SocketAddress;Lio/
    netty/channel/ChannelPromise;)Lio/netty/channel/ChannelFuture;+7
j  io.netty.channel.AbstractChannel.connect(Ljava/net/SocketAddress;Ljava/net/SocketAddress;Lio/netty/
    channel/ChannelPromise;)Lio/netty/channel/ChannelFuture;+7
j  io.netty.bootstrap.Bootstrap$2.run()V+56
j  io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(J)Z+26
j  io.netty.channel.nio.NioEventLoop.run()V+106
j  io.netty.util.concurrent.SingleThreadEventExecutor$2.run()V+13
j  io.netty.util.concurrent.DefaultThreadFactory$DefaultRunnableDecorator.run()V+4
j  java.lang.Thread.run()V+11
v  ~StubRoutines::call_stub
V  [libjvm.dylib+0x2db378]  JavaCalls::call_helper(JavaValue*, methodHandle*, JavaCallArguments*,
    Thread*)+0x22a
V  [libjvm.dylib+0x2db88f]  JavaCalls::call_virtual(JavaValue*, KlassHandle, Symbol*, Symbol*,
    JavaCallArguments*, Thread*)+0x11b
V  [libjvm.dylib+0x2db9cc]  JavaCalls::call_virtual(JavaValue*, Handle, KlassHandle, Symbol*, Symbol*,
    Thread*)+0x4a
V  [libjvm.dylib+0x32a89e]  thread_entry(JavaThread*, Thread*)+0xad
V  [libjvm.dylib+0x4f5f69]  JavaThread::thread_main_inner()+0x9b
V  [libjvm.dylib+0x4f7671]  JavaThread::run()+0x1a3
V  [libjvm.dylib+0x420e02]  java_start(Thread*)+0x126
C  [libsystem_pthread.dylib+0x3268]  _pthread_body+0x83
C  [libsystem_pthread.dylib+0x31e5]  _pthread_body+0x0
C  [libsystem_pthread.dylib+0x141d]  thread_start+0xd

(processes and system details skipped)
```

Listing A.1: Exception when using deserialized crippled IPv6

Appendix B

Additional results

B.1 Idle Energy Consumption

	Run 1	Run 2	Run 3	Average
TCP channel	633 mW	608 mW	613 mW	618±13 mW
No TCP channel	617 mW	580 mW	598 mW	598±18 mW
Savings	2.47%	4.50%	2.52%	3.16%

Table B.1: The average idle energy consumption of three measurement runs in Wifi with a routing table update interval of 120s. Each measurement run was conducted at a different access point. The last row shows the savings of having no open TCP channel. The average savings of omitting the open TCP channel in Wifi is only 3.16%.

	O_2	Sunrise	Swisscom	Average
TCP channel	1029 mW	1259 mW	1034 mW	1107±132 mW
No TCP channel	698 mW	721 mW	691 mW	703±16 mW
Savings	32.16%	42.75%	33.16%	36.02%

Table B.2: The average energy consumption of a connected mobile peer in 3G measured with three different cellular network providers. The routing table update interval is 120s. In average, more than 36% of the energy is saved when not having an open TCP channel in a 3G network.

B.2 Hardware Measurements

B.2.1 Wifi

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$
Full peer	662 mW	198 s	130 J
Reverse connection relaying	723 mW	198 s	142 J
Buffered direct relaying (20s)	672 mW	214 s	142 J
Buffered direct relaying (40s)	627 mW	208 s	129 J
Buffered direct relaying (60s)	613 mW	247 s	150 J
BPN relaying (20s)	659 mW	214 s	139 J
BPN relaying (40s)	597 mW	210 s	124 J
BPN relaying (60s)	608 mW	247 s	149 J

Table B.3: Measurement 1 in Wifi

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$
Reverse connection relaying	701 mW	198 s	139 J
Buffered direct relaying (20s)	641 mW	214 s	136 J
Buffered direct relaying (40s)	623 mW	210 s	130 J
Buffered direct relaying (60s)	642 mW	247 s	157 J
BPN relaying (20s)	663 mW	212 s	138 J
BPN relaying (40s)	662 mW	206 s	136 J
BPN relaying (60s)	658 mW	242 s	158 J

Table B.4: Measurement 2 in Wifi

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$
Reverse connection relaying	658 mW	198 s	129 J
Buffered direct relaying (20s)	655 mW	210 s	136 J
Buffered direct relaying (40s)	642 mW	208 s	132 J
Buffered direct relaying (60s)	645 mW	247 s	158 J
BPN relaying (20s)	618 mW	214 s	131 J
BPN relaying (40s)	600 mW	210 s	125 J
BPN relaying (60s)	608 mW	247 s	148 J

Table B.5: Measurement 3 in Wifi

B.2.2 3G

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$	$\text{Savings}_{\text{avg}}$	$\text{Savings}_{\text{total}}$
Reverse connection relaying	1290 mW	198 s	253 J	0%	0%
BPN (20s)	894 mW	199 s	177 J	30.70%	30.04%
BPN relaying (40s)	800 mW	211 s	171 J	38.02%	32.41%
BPN relaying (60s)	784 mW	246 s	191 J	39.21%	24.51%

Table B.6: Measurement with Swisscom in 3G

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$	$\text{Savings}_{\text{avg}}$	$\text{Savings}_{\text{total}}$
Reverse connection relaying	1167 mW	199 s	230 J	0%	0%
BPN relaying (20s)	990 mW	198 s	194 J	15.24%	15.65%
BPN relaying (40s)	832 mW	215 s	177 J	28.71%	23.04%
BPN relaying (60s)	822 mW	247 s	201 J	29.55%	12.61%

Table B.7: Measurement with Sunrise in 3G

	$\text{Power}_{\text{avg}}$	Time	$\text{Power}_{\text{total}}$	$\text{Savings}_{\text{avg}}$	$\text{Savings}_{\text{total}}$
Reverse connection relaying	1312 mW	198 s	256 J	0%	0%
BPN relaying (20s)	770 mW	235 s	179 J	31.27%	30.08%
BPN relaying (40s)	739 mW	214 s	157 J	43.65%	38.67%
BPN relaying (60s)	740 mW	246 s	181 J	33.62%	29.30%

Table B.8: Measurement with O_2 in 3G

B.3 Software Measurements

The charts presented next are software measurements. Parts of the source code of PowerTutor has been integrated into the measurement application in order to estimate the energy consumption of the DHT interaction. The advantage is that the energy consumption can be estimated for each process on the mobile phone individually. Therefore, the following plots do not show the total mobile phone's battery drain, but the energy consumption of the network adapter as if it was only used by the test application. The disadvantage of software measurements is their inaccuracy. The used power estimation model is from an outdated phone model (HTC Dream), applied to a Samsumg Galaxy S II [24].

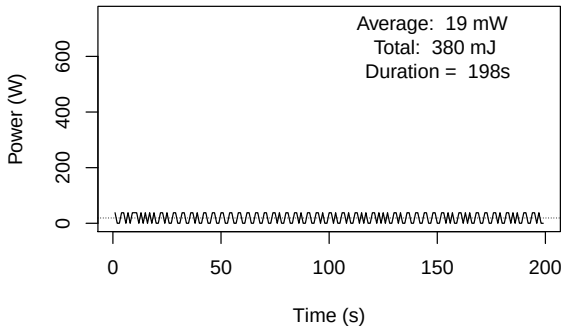
The buffered direct relaying is not shown because the findings indicated that this method does not have any advantage over reverse connection relaying or BPN relaying. The energy levels of the spikes are all the same because the estimation application uses fixed numbers for each network adapter state. The charts therefore rather show the current network adapter state than the real energy consumption.

B.3.1 Wifi

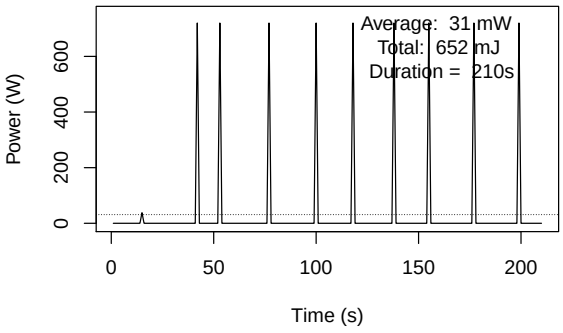
Figure B.1 shows a comparison of the energy consumption with Wifi connectivity. The used energy estimation model quantifies the transmit_L with 39 mW and transmit_H with 720 mW. If no data is transmitted, it is assumed that the adapter does not consume any battery (i.e. 0 mW). Thus, each network interaction represents a spike of either height 39 mW or 720 mW. The difference between the non-buffered reverse connection relaying and the BPN approach is clearly visible. Since reverse connection relaying only needs to answer a single put request, switching to transmit_L suffices. Each time the buffer is obtained and answered in BPN relaying, the amount of traffic exceeds the threshold for switching to transmit_H, requiring more energy.

B.3.2 3G

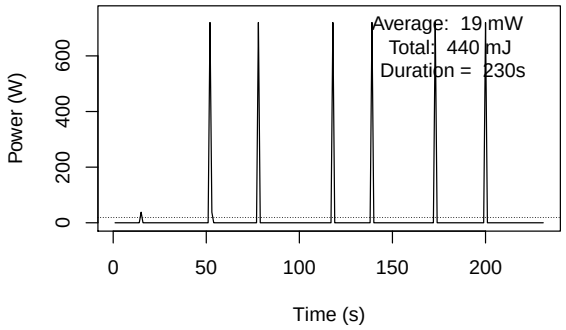
Figure B.2 shows a comparison of the energy consumption with 3G connectivity. Again, only the energy consumption of the network adapter is shown, whereas the constant values for the 3G power model are 401 mW for the FACH and 570 mW for the DCH state. If the 3G network adapter remains in the IDLE state, an energy consumption of 10 mW is assumed. Frequent arriving requests in the reverse connection relaying prevent the network adapter from demoting to the IDLE state. The steadily recurring traffic even forces it to remain in the DCH mode. Energy can be saved using BPN relaying because the 3G network adapter can switch back to the IDLE mode between the buffer timeouts. Compared to Wifi, the spikes in BPN relaying are wider because the tail-times of the 3G power state machine are higher.



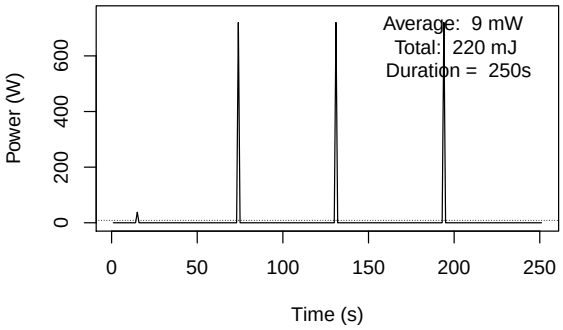
(a) Reverse connection relaying (non-buffered)



(b) BPN relaying (20s buffer)

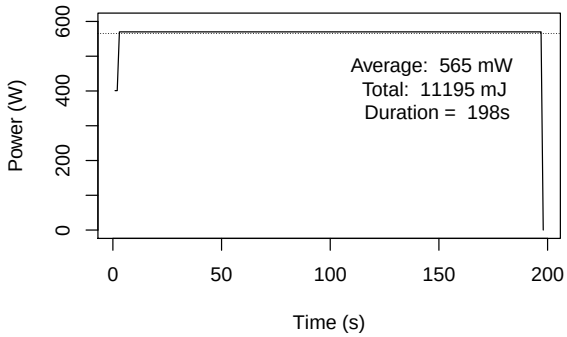


(c) BPN relaying (40s buffer)

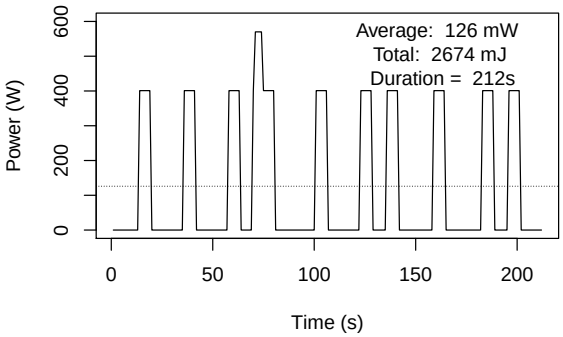


(d) BPN relaying (60s buffer)

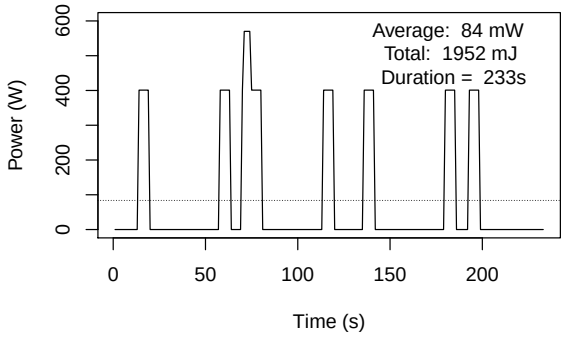
Figure B.1: Software measurements in Wifi



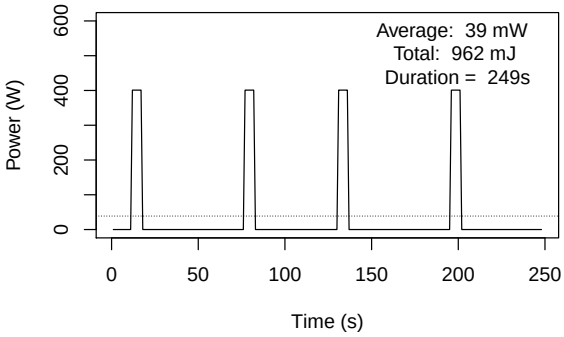
(a) Reverse connection relaying (non-buffered)



(b) BPN relaying (20s buffer)



(c) BPN relaying (40s buffer)



(d) BPN relaying (60s buffer)

Figure B.2: Software measurements in 3G

Appendix C

Contents of the CD

App:

Icons - Contains icons and logos used in the application.

org.hive2hive.mobile.zip - The source code for the Android application.

org.hive2hive.mobile.apk - The packaged Android app.

Screenshots - Screenshots of the Android app.

Evaluation:

Data - All files from the software and hardware measurements and analysis scripts.

Sources - The source code for the measurement application on the mobile phone and the fixed-network peers.

Thesis:

Abstract.txt - A copy of the abstract.

Presentation.pdf - A copy of the midterm presentation.

RelatedWork.zip - Copies of related work papers.

Roadmap.pdf - The roadmap used for this thesis.

Thesis.pdf - A copy of the written thesis in PDF format.

Thesis.ps - A copy of the written thesis in PS format.

Thesis.zip - Source files of the written thesis.

Zusfg.txt - A German version of the abstract.