EAI Endorsed Transactions Preprint Research Article

Analysis of Differential Synchronisation's Energy Consumption on Mobile Devices

Jörg Simon[1,*], Peter Schmidt[2], Viktoria Pammer-Schindler[3]

[1]Know-Center GmbH, Inffeldgasse 13/6, 8010 Graz, Austria
[2]Mendeley Ltd, White Bear Yard 144a Clerkenwell Road, London, UK
[3]Knowledge Technologies Institute, Graz University of Technology, Inffeldgasse 13/5, 8010 Graz, Austria

Abstract

Synchronisation algorithms are central to collaborative editing software. As collaboration is increasingly mediated by mobile devices, the energy efficiency for such algorithms is interest to a wide community of application developers. In this paper we explore the differential synchronisation (diffsync) algorithm with respect to energy consumption on mobile devices. Discussions within this paper are based on real usage data of PDF annotations via the Mendeley iOS app, which requires realtime synchronisation.
We identify three areas for optimising diffsync: a.) Empty cycles in which no changes need to be processed b.) tail energy by adapting cycle intervals and c.) computational complexity. Following these considerations, we propose a push-based diffsync strategy in which synchronisation cycles are triggered when a device connects to the network or when a device is notified of changes.

Received on XXXX; accepted on XXXX; published on XXXX

Keywords: synchronisation, collaboration, differential synchronisation, energy efficiency, mobile computing, push notification mechanism

doi:10.4108/XX.X.X.XX

1. Introduction

Enabling synchronous read and write access to shared items is one of the key functionalities in collaborative software (e.g., svn), games (e.g, Age Of Empires [2]) or documents (e.g., Google Docs). In the wake of increased mobile broadband access distributed collaboration is increasingly mediated by mobile devices. Application developers should, therefore, consider energy consumption in the use of synchronisation algorithms.
In this paper we explore the energy consumption of the differential synchronisation algorithm (diffsync) from the viewpoint of mobile devices. Diffsync is a synchronisation algorithm which is originally designed for near real time synchronisation over the web, like online computer games or collaborative document editing. We showed that is it possible to configure the diffsync algorithm to use push notifications and with it retaining the correctness of the algorithm and saving energy at the same time still keeping the near realtime nature of the algorithm in tact [22]. In this paper we want to explore the energy characteristics in more detail.
A preliminary version of this paper was published as a poster at the Mobiquitous 2014 [22]. This paper extends the poster by an in-depth analysis of the energy characteristics of different cycle times, the energy characteristics of size and complexity of single diffs on the CPU, describe the experiment setup and use case in more detail in order to give readers the possibility to repeat similar experiments, and also go more in depth on how the push notifications work and save energy.

2. Use Case: Social Reference Management

Examples in this paper are based on real usage data of editing PDF annotations within the Mendeley iOS app. Mendeley is a social document and content reference management tool, e.g. it enables users to manage their PDF documents, generate citations and bibliographies. In the Mendeley iOS app, PDF content can be annotated, e.g., highlighting, or adding and

*Corresponding author email: jsimon@know-center.at

editing textual notes. Together with other metadata, annotations are stored locally on the mobile device as well as centrally on the Mendeley server. References, and their related PDFs can be shared amongst a group, and PDF annotations can therefore be edited by multiple users. Data integrity between server and Mendeley app (client) are a fundamental requirement. This raises the need for synchronising data between client(s) and server. Since annotations can be done collaboratively, diffsync was a natural choice for mendeley, contrary to simpler direct synchronisation schemes or rsync schemes services like dropbox use it. Mendeley used the diffsync algorithm in their official iOS client app during the time of the study. We can use this data to report realistic usage schemes and package sizes for later calculations.

3. Related Work

3.1. Energy Optimisation on Mobile Devices

Mobile devices such as smartphones and tablets rely on battery power. Energy intensive hardware and software operations limit the use by "shortening battery life" and significantly impact user experience. Energy consumption can be reduced either by modifying mobile device hardware or mobile applications (software). In this paper, we are concerned with the latter. For application developers the challenge is to balance the need for performance (e.g. computational speed) while maintaining a low level of power consumption. For instance, in mobile phone sensing, it is common to reduce sampling frequency of sensors to reduce energy consumption. This may lead to reduced precision however [20, 21]. Moreover, the most energy consuming components in mobile devices are CPU, network components (WiFi, 3G, GSM) and display [4, 5][1]. As optimising display energy consumption is out of application developers' control except in game development, application developers are mostly left to optimise CPU and network energy consumption. In [1], a network energy consumption model is described based on empirical data, which emphasises the possibility to optimise network energy consumption by exploiting the tail energy property of different network protocols. This model has influenced other research e.g. for optimising computational complexity on the mobile device by offloading computation to the cloud [6, 12]. Network energy consumption can sometimes be optimised by choosing suitably between push and pull communication paradigm, as has been discussed in [3] in connection with the Google cloud messaging system. CPU energy consumption can be optimised by adapting algorithms to mobile devices, as has been done e.g., for the hash function [7] and AdaBoost [14].

[1] https://source.android.com/devices/tech/power.html

3.2. Delay Tolerant or Prefetch Friendly Synchronisation

Delay tolerant or prefetch friendly synchronisation methods can drop the constrain that small updates should be delivered frequently. As we will see, in a scenario where frequent small updates must be made, tail energy is the main concern. The best strategy to save energy is to collect a bulk of data for transmission for a while, and transmit at a time of good connectivity (of fetch a lot respectively). This minimises the tail energy, as this is a constant added only once after the bulk transmission. Is also minimises transmission energy, as this is inverse proportional to the bandwidth available. Thus, using time slots of good connectivity is an important part of Mobile Cloud Computing [9]. Recent research formulates this problem as a system of applications putting data into queues for transmission, a discrete time, and a cost function based on a power model. A Lyapunov drift is then used to compute when data should be transmitted. AppATP [10] is a cloud based middleware managing several applications, and decides for prefetch friendly and delay tolerant apps in a fixed cycle of 60s if a transmission should be made. AppATP archives savings of 30- 50% on the synchronisation part of a complete system. eTrain [24] also uses Lyapunov optimisation, and also optimises several mobile Applications at once. However, it does not rely on a own middleware, and piggyback the heartbeat transmissions applications employing push notifications need to keep the tunnel open to add transmissions there to reduce wasted tail energy. It can save 12-33%. Similar to both Systems our approach uses a fixed cycle time. Similar to eTrain reducing the waste of energy caused by tail energy is a central topic. However, we want to keep the near real time character of the diffsync. Therefore we employ a different strategy for tail energy (optimising cycle time/tail energy tradeoff and push notifications). Also these works look at group of apps while we look at suggestions for an specific algorithm for an app developer to implement.

3.3. Realtime Synchronisation

For a long time, operational transform (OT) [8] has been the quasi-standard synchronisation algorithm in use. It is used for instance in Google products like Google Docs or Google Wave. OT is based on the notion of expressing edits as operations on the state of an item (e.g. a document). To avoid loss of changes and ensure data integrity, OT relies on capturing all user edits for each item. Once an edit is lost, different item copies will not converge towards an agreed version again. This poses a significant challenge given today's rich

user interfaces and feature sets, including: edit items via typing, cutting, pasting, undo, redo, drag, drop, etc. [11][2].

There is an "intent" problem with OT [23]: It may happen, that a correct merge of two edits is in conflict with the intent of each individual edit. E.g. let us assume that two users edit the sequence “AB”, one person wishes to delete “B”, and the other to insert “12” after the “A”. An intent-preserving merge would lead to “A12”. Depending on the sequence of transformation operations, however, OT may lead to “A1B”, which does not correspond to the intent of either editor. Woot [19] (WithOut OT) tries to resolve this problem by including not only the exact transformation operation but also contextual information in one operation. However, Woot does not allow deleting content. A successor, Logoot [25] allows deletions as well as undos [26].

Differential Synchronization (diffsync) [11] works on item states only - i.e. knowledge of edit actions is not required by the synchronisation algorithm. The key notion of diffsync is that the synchronisation algorithm compares two states of an item, computes the differences and necessary changes. While some changes may be lost, different item copies will always converge. Diffsync is used in the Mendeley iOS app for synchronising PDF annotations, but also in MobWrite[3] for collaborative writing or CoRED [15] for collaborative programming. More widespread use of diffsync can be expected in the future, mainly because of the ease of implementation with fewer possibilities for programming errors when compared to other synchronisation algorithms. Firstly, client and server code are nearly identical. Secondly, synchronisation code does not depend on user interface code. The latter is highly relevant in collaboration systems available on a broad range of devices. Thirdly, synchronisation code is independent of the diff and patch algorithm, which makes synchronisation code independent of the data structure of an application. Thus, diffsync delegates the problem of preserving edit intentions to the diff and patch algorithm. It is therefore “suitable for any content for which semantic diff and patch algorithms exist” [11].

Causal Trees [13] mix features of Woot and diffsync, but break the elegant independence between synchronisation and diff/patch algorithm that exists in diffsync.

4. Differential Synchronization

Diffsync for one client and one server can be briefly summarised as follows: Both server and client contain an item (e.g. a document) on which the diffsync algorithm will be executed. As will be shown below, both server and client will also create a "shadow" copy of the item. The initial state is defined as having no edits and assumes that all copies of the item, including the shadow copies, are identical. The following list describes one diffsyc cycle starting with an edit on a client item. Figure 1 illustrates this procedure.

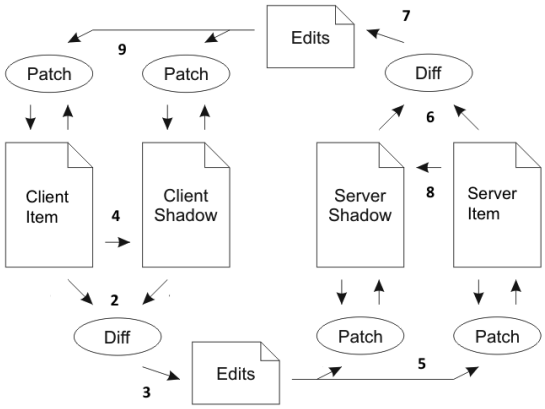

Figure 1. Differential synchronization with shadows – essentially corresponds to Fig.3 in [11] but the numbering and text have been adapted.

1. User edits client item.
2. Diff: Difference between client item and client shadow is computed.
3. A list of necessary edits is the result.
4. Client item is copied to client shadow.
5. Patch: The edits are patched onto the server shadow (which was identical to the client shadow until Step 4) and onto the server item (which may be different to the server shadow, e.g., in a multi-device/multi-user environment).
6. Diff: Difference between server shadow and server item is computed.
7. A list of necessary edits is the result (edits may be due to unsuccessful take-up of client edits, or due to edits from other devices/users).
8. Server item is copied to server shadow.
9. Patch: The edits are patched onto the client shadow (after applying the patch, the client shadow is identical to the server shadow) and onto the client item.

This algorithm can be extrapolated to a system where an item is shared between multiple clients: The server maintains a central copy of the item (server item) and a shadow copy for each client (multiple server shadows).

[2]See also http://sharejs.org.
[3]https://code.google.com/p/google-mobwrite/

5. Potential for Energy Optimisation

There are three areas with potential for energy optimisation. All three are based on the underlying property of the unmodified differential synchronisation that the complete diffsync cycle as described above (Steps 1-9) is executed in regular, fixed intervals.

5.1. Empty Cycles

The unmodified diffsync executes the complete diffsync cycles in regular, fixed intervals even if there are no changes (empty cycles). Clearly, these cycles consume CPU and network (3G, WiFi, Bluetooth) energy without direct benefit to synchronisation.

5.2. Tail Energy

In 3G and GSM connections, there is a medium power state lasting for approximately $12s$ for 3G and $6s$ for GSM once a data connection is terminated. This tail energy accounts for up to 60% of the energy consumption of a network connection [1]. If you transfer at fixed cycle intervals it is therefore more efficient to either transfer data more frequently (more than $12s$ resp. $6s$), or to transfer large amounts of data in less frequent intervals. Therefore, the worst interval cycle for diffsync would be $12s$ or $6s$ depending on connection type: The mobile device would not be able to enter a low power state (network sleep). This leads to wasting energy in the medium power (tail energy) state, offsetting computational advantages by processing very small changes (see next discussion item). Note that for WLAN connections, there is no such tail energy [1], and that different sources quote slightly different durations of power states[4]. Also note that for synchronisation that is delay tolerant it is more efficient to basically collect transmissions until a large chunk of data is available and transmit at a time with good connectivity [10, 16, 24].

5.3. Computational Complexity

Computational complexity directly correlates with CPU energy consumption [7]. Small changes have a high likelihood of being computationally less complex than large changes [17]. In addition, item size influences computational complexity, as diff is of $O(n^2)$ [17].

This discussion leads to the following conclusions: a.) repeated diffsync cycles containing no changes, i.e. "empty" cycles, should be avoided. b.) to optimise tail energy consumption, the recommended cycle interval is below $12s$ for 3G (resp. $6s$ for GSM), or considerably longer interval durations (than $12s$). Finally c.) processing small changes and small items reduces computational complexity and reduces energy consumption. If cycle times can be considerably longer, Lyapunov optimisation should be considered.

5.4. Experimental Verification Based on the Mendeley App

We verified the effects of tail energy and computational complexity on energy consumption in simulations with the iOS Mendeley App.

Measuring Energy Consumption on Mobile Devices Many papers report power consumption of components in W (Watt) and in J (Joule, $1J = 1Ws$) of events, as these are the correct physical terms. In order to compute power consumption in terms of W or J, system voltage of the battery would need to be measured over time. Since we are only interested in the relative energy consumption of different configurations, we have chosen to measure energy consumption in terms of percent of battery power. We did this via a software monitor of battery status, using an iOS internal API which reports battery drainage in 1% steps. This call requires no significant energy in case the CPU is already active. A similar setup to measure energy consumption of apps has been described in [18] for both Android and iOS. We therefore express energy consumption in terms of battery drainage in % per minute ($\%/min$) in this paper.

Experiment Setup We have implemented an experimental framework which includes the software monitor of battery status and an experiment runner. The experiment runner starts executions of the diffsync algorithm as implemented within the Mendeley app but requires no user interaction. The runner takes a measurement of battery status before and after each diffsync cycle. Server-side action in the experiments has been simulated based on real usage data from Mendeley. With that method more than 60.000 diffsync cycles were recorded, with about 1.000 cycles per experiment.
All our experiments run on the same device, an iPhone 5s with a fresh install of iOS7. The following device setup was used: iCloud was disabled, one email account with manual updates, no notifications, no iMessage and no other apps apart from default system apps. In every experiment, unused network components were switched off in order to avoid battery drainage through low power states or network scans. The experiment runner dims the screen to minimal brightness before each experiment run.
The baseline drainage of the environment without execution of diffsync is 0.108% of battery power per

[4]For instance, an additional high power state of $5s$ duration is included in the network model at http://developer.android.com/training/efficient-downloads/efficient-network-access.html.

minute. This has been averaged over a total of 10h recording with sampling time for battery measurement ranging from 1s to 60s.

Tail Energy Consumption Based on existing models of network energy consumption [1], we have predicted above that in order to optimise tail energy, the diffsync cycle interval should either be significantly below 12s (3G) or 6s (GSM), or significantly above. For WLAN, we expect no dependance on cycle interval.
In order to verify this, the experiment runner executed diffsync with different cycle intervals for 3G, GSM and WLAN network connection: The cycle intervals were 0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 20, 30, 60 seconds. The downlink bandwidths were 168,2 kbit/s for GSM, 2,4 mbit/s for 3G, and 2,8 mbit/s for WLAN averaged over all runs. We observed that a usual package is less than 1kb in our case, and most of the time about 124$bytes$ in diffsync with a 3s cycle interval. So, for every package size the transfer was always less than a second, and almost all the energy has to account to cycle time and cpu complexity. Since we were interested in cycle time energy without diff energy, and the difference between the standard package (124$bytes$) and the empty package (100$byte$) are not significant for this test, we choose to send empty packages around. Each experiment was run until 20% of the battery was drained. The dependance of diffsync's energy consumption on the cycle interval is illustrated in Fig. 2.
Experiments show that the predictions based on the network energy consumption model of [1] are correct. Additionally, we can experimentally identify a local minimum around a cycle interval of 6s for 3G. In case a fixed and small cycle interval is desired, 6s would then be a recommended interval for all network protocols: For WLAN, the interval does not matter, for 3G it is the local minimum, and GSM does not use much less power even for smaller cycle intervals.

Effect of Computational Complexity on Energy Consumption Theory predicts that computationally more complex changes require more CPU time and, therefore, consume more energy. Computational complexity for diffsync stems from the complexity of the change, and from the item size.
We verified and quantified this in an experiment: The experiment runner executed diffsync with a cycle interval of 6s. In the first run, all cycles were empty, i.e. no changes were made. In the second run, in each cycle a worst case (from a computational complexity viewpoint) change happened on an item of size $\approx$ 700$bytes$. In the third run, in each cycle a worst case change happened on an item of size $\approx$ 7000$bytes$. In a fourth run, a "simple" change happened in every cycle

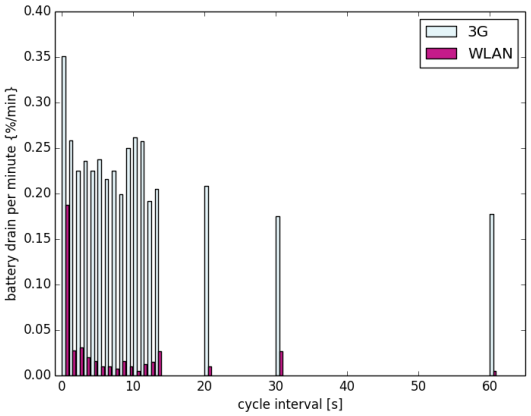

Figure 2. Comparison of battery drainage of 3G vs. WLAN, based on cycle timing

on a large item ($\approx$ 7000$bytes$). In all four experiment runs, the experiment was run until 20% of the battery was drained.
When the data structure is an ordered list, the worst case change is a change in both the beginning and the end of the list for a fixed size of the change. For instance, if every change has the size of two characters, it is computationally more complex if "cat" is changed to "bad" than if "cat" is changed to "colt". Ordered lists are a typical data structure for text, as is the case for PDF text annotations in Mendeley. A simple change is for instance a deletion at the end of the item, e.g., "cat" is changed to "ca".
In the experiment run with empty cycles, energy consumption was 0.22%/min. In the experiment run with the computational worst case and small item size of 700$bytes$ as well as in the experiment with a simple change but large item size of 7000$bytes$, no difference in energy consumption when compared to empty cycles could be measured via the software monitor. Only for a very large item size of 7000$bytes$ and the worst case change, energy consumption was significantly, namely 0.34%/min.
This confirms the prediction that computational complexity impacts energy consumption, but only when item size is large and the change is complex. Since developers can influence item size but not complexity of changes (this lies with the users), energy consumption due to computational complexity can best be avoided by reducing item size. This argues for data structures that break content down into small pieces, so that diff and patch algorithms can work on small items. For instance, documents should be broken down into paragraphs or sections.

Generalisation of Experiment Results The experiments are performed on iOS, on an iPhone 5s, and on

a single device. Results can nonetheless be generalised to other iPhone 5s devices, other generations of iPhones, as well as to other platforms (Android, Windows 8) and phones from different manufacturers: Energy savings are either specific to the network protocol, like tail energy to GSM and 3G, or to the CPU. While there are differences in Apple's Push Notification Service and Google's Cloud Messaging, they rely on the same principles with respect to network energy consumption [3]. Both platforms use ARM multiprocessors, and provide similar advice in terms of optimising for CPU energy efficiency[5]. The remaining difference is the operating system: By switching off as many services as possible in order to still allow the Mendeley app to be executed, we avoid measuring operating system services such as background download of emails etc. Apart from this, the push-based diffsync optimisation works on an algorithmic level, without platform-specific elements.

6. Push-Based Energy Optimisation of Differential Synchronisation

The original diffsync paper [11] suggests adapting cycle intervals to current editing activities in a client between 1*s* and 10*s*. However, this has been proposed in view of improving performance (e.g. processing speed, avoidance of merge conflicts), rather than energy consumption.

In order to strike a balance between computational performance and power consumption we propose to execute a diffsync only when changes occur, except for an initialisation cycle. Concretely:

1. In the initial state, the client connects to the network. In order to capture any changes that may have occurred in the meantime an initial diffsync cycle will be required.
2. When the client item is edited, the client initiates a diffsync cycle.
3. If a change arrives at the server, a push notification is sent to all clients. On receiving the notification clients execute a complete diffsync cycle.

Nearly No Empty Cycles Except at initialisation, no empty cycles are being carried out. This is the central property of the push-based optimisation of energy efficiency of diffsync. Note that reducing empty cycles only reduces energy consumption significantly for 3G and GSM connections, as empty cycles in WLAN drain the battery only minimally.

Correctness The push-based optimisation algorithm does not change the diffsync algorithm per se, but only changes the intervals between cycles. All edits are synchronised to the server and to connected clients as fast as possible.
Clients not connected to the network (offline) cannot receive push notifications. In this case clients may have to process significant amount of changes when reconnecting to the network (online). Depending on the complexity and nature of changes this may lead to merge conflicts and failures. However, this is also the case for the original diffsync.

Identifying the Occurrence of an Edit The original diffsync algorithm is, amongst other things, easy to implement because as it is independent of the nature of "an edit". The algorithm only needs to compare states regularly. In contrast, the push-based optimisation algorithm needs to know what an edit is in Step 2. Note that also Fraser's suggestion for bounded adaptive timing [11] would require this knowledge.
There are three possibilities how to identify that an edit has occurred on the client: Firstly, the "diff" part of the diffsync algorithm could be executed in regular intervals until a difference (an edit) is detected. Secondly, the data structure (e.g., file, database, in-memory) in which the application stores its content can be monitored for changes. Thirdly, it can be decided which user interaction means that an edit has occurred. Depending on the application and which granularity of edits needs to be identified, this may be as simple as noticing a user pushing a "save" button, or as complex as tracking every possible way of editing within a given UI.
Note that the difficulty (for implementation) of identifying an edit is lower than in operational transformation (OT). While in the push-based optimised diffsync, one only needs to identify that an edit has occurred, OT requires exact knowledge of what the edit consists of in addition.

Energy Consumption of Push Notifications on Mobile Devices A network communication overhead is generated via push notifications. In both Android and iOS, push notifications are facilitated by a local service (on the mobile device) which regularly polls for push notifications (Google cloud messaging[6], Apple push notifications[7]).
We quantified this overhead in an experiment: The local

[5] For Apple see https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/PerformanceTuning/PerformanceTuning.html, for Android see http://developer.android.com/training/monitoring-device-state/index.html

[6] http://developer.android.com/google/gcm/index.html
[7] https://developer.apple.com/notifications/

service that polls for push notifications was run for $2h$ without incoming push notifications (idle polling). Idle polling adds $\approx 0.02\%/min$ of energy consumption to the idle operating system, thus adding nearly no overhead. In a $2h$ run with a push notification every $6s$ (active polling), energy consumption was $0.13\%/min$. This overhead is only for receiving push notifications, not for executing the diffsync cycles. For instance, if every $6s$ a change on a small item happens, overall energy consumption would be $\approx 0.35\%/min$. This number stems from adding the energy consumption of active polling with notifications every $6s$ ($0.13\%/min$) to the energy consumption of diffcycles every $6s$ for items of small size ($0.22\%/min$ for items of size of $\approx 700bytes$).
Therefore, the polling required on mobile devices in case of a push notification mechanism leads to a negligible energy consumption overhead while no changes occur (idle polling). This means that the push notification mechanism nearly eliminates the energy wasted in empty cycles in the original diffsync. However, active polling introduces an overhead of $\approx 0.13\%/min$. This means that if changes occur regularly and frequently, it is more energy efficient to adapt cycle time to editing activity.

Minimum Time Between Push Notifications Note that when the interval between push notifications becomes smaller than $2s$, the mechanism became highly unreliable in our experiments, in the sense that the sequence of notifications was changed or notifications became lost. Server code should therefore take care to send notifications about changes only in intervals larger than $2s$. In addition, clients need to ensure that only one diffcycle is running at a time.

7. Push-Based Diffsync within Mendeley

We have implemented the above-described push-based energy optimisation of diffsync in the Mendeley app. In this section, we describe our implementation, emphasising app specifics in terms of implementation and usage that impact the actual effect of this optimisation.

7.1. Diffsync in the Mendeley App

In the Mendeley app, diffsync is active in clients when a PDF document is opened, and inactive when the document is closed. Synchronisation is on PDF annotations such as textual highlights and notes. The data content exchanged in each cycle is based on differences in position, and colour for highlights. When adding and managing notes, differences in text and author names as well as position and colour will be added. Therefore, highlighting text results in a smaller diffsync payload than annotation notes. The payload also includes other synchronisation metadata, e.g. unique identifiers. The format of the exchanged data set is based on standard JSON.
The Mendeley app's internal data structure for PDF annotations is a dictionary where every PDF annotation (highlight or note) is a dictionary entry identified by a unique ID. The "diff"-part of the diffsync algorithm therefore works on very small items, namely single dictionary entries. Thus, computational complexity has no significant effect on diffsync energy consumption in the Mendeley app. Data are stored in a Core Data[8] database.

Original Diffsync The diffsync cycle interval in the implementation of the original diffsync was $2s$ before the push-based optimisation. In empty cycles, approximately $100bytes$ of data (metadata) are transmitted.

Push-based Diffsync For correctness, we ensure that only one diffsync cycle can run at a time. If a push notification arrives or a local edit is identified while a diffsync cycle is running, we let the algorithm perform another cycle right after the current one. We do so via a boolean flag `loopOnceMore`. When diffsync starts a cycle, the flag is set to `false`. In case a push notification arrives while a cycle is running, the flag will be set to `true`. At the end of each cycle diffsync checks that flag, and if `true` performs another cycle.
We identify the occurrence of edits on the client by listening to modifications of the data structure. Since CoreData sends notifications about changes to its content in any case, this does not add extra computation and therefore does not consume additional energy.
Servers-side, push notifications are sent to all clients, but waits at least $2s$ between sending push notifications in order to avoid unreliable behaviour.

7.2. Mendeley Usage Statistics

We analysed Mendeley usage data from a period of 4 weeks in May 2014. In this period, users were far more likely (with an approximate ratio of $3:1$) to change text highlights than to edit textual notes. This strongly supports the conclusion that computational complexity does not offer potential for energy optimisation in the Mendeley app, since item sizes are very small not only by design but also by usage.

Average Interaction Time with PDFs On average, users spent $431,4s$ ($\approx 7min$) within a PDF, reading and

[8] https://developer.apple.com/library/iOS/documentation/Cocoa/Conceptual/CoreData/

editing. Of course, the interaction time varies widely, with a minimum of $3,3s$ and a maximum duration of $1977,8s$ ($\approx 33min$).

Empty Cycles A total of 100.000 diffsync cycles were analysed from the server logs. Of these, 96792 were empty cycles. Thus, an overwhelming majority of diffsync cycles (96.8%) are empty and waste energy.

7.3. Reduction of Energy Consumption via Push-Based Diffsync in Mendeley

96.8% of diffsync cycles in the non-optimised Mendeley app are empty, thereby wasting energy. The push-based optimisation of diffsync therefore significantly reduces energy consumption. We can quantify this, using average use data from Mendeley usage statistics. Assuming, a user stays $7min$: in a document, and on average 96.8% of cycles of the original diffsync are empty, and the user has a 3G connection. The original diffsync with a $2s$ cycle interval drains the battery 1.729% in these $7min$. The push-based diffsync however drains the battery only 0.084% in these $7min$. For the maximum time in a PDF observed in the usage data we analysed, $33min$, this looks even more drastic: The original diffsync drains $8,151\%$ of battery power, while the push-based diffsync uses only 0.398% of battery power.

8. Conclusion

Differential synchronisation as realtime synchronisation algorithm for collaborative editing systems has three potential areas for optimising energy consumption: Empty cycles, tail energy (cycle intervals) and computational complexity. We have shown theoretically, and verified in experiments, that a.) tail energy optimisation argues for cycle intervals of $\approx 6s$ for 3G and b.) the impact of computational complexity on energy consumption can best be addressed by appropriate data structures that organise content into small items. Tail energy optimisation is useful for instance when server code cannot be changed, as the cycle interval is determined by the client.

Most significantly, we have proposed the push-based optimisation of differential synchronisation, which eliminates empty cycles (except for initialisation purposes). This optimisation is useful in collaborative editing systems where edits are rather infrequent, as is the case in the Mendeley app which we used to showcase the benefits of the push-based optimisation. In such systems, the push-based optimisation of differential synchronisation leads to a system with both higher response time and lower energy consumption.

Overall, we emphasise, that energy optimisations of differential synchronisation should be done based on knowledge of a collaborative system's usage data.

9. Acknowledgments

The Know-Center is funded within the Austrian COMET Program under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

References

[1] Balasubramanian, N., Balasubramanian, A. and Venkataramani, A. [2009], Energy consumption in mobile phones: a measurement study and implications for network applications, *in* 'Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09', ACM Press, New York, New York, USA, pp. 280–293.

[2] Bettner, P. and Terrano, M. [2001], 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond, *in* 'Proceedings of the 15th Games Developers Conference - GCD '01', San Jose, CA, USA.

[3] Burgstahler, D., Lampe, U., Richerzhagen, N. and Steinmetz, R. [2013], Push vs. Pull: An Energy Perspective (Short Paper), *in* 'Proceedings of the 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications', SOCA '13, IEEE Computer Society, Washington, DC, USA, pp. 190–193.

[4] Carroll, A. and Heiser, G. [2010], An Analysis of Power Consumption in a Smartphone, *in* 'Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference', USENIXATC'10, USENIX Association, Berkeley, CA, USA, pp. 21–21.

[5] Crk, I., Albinali, F., Gniady, C. and Hartman, J. [2009], Understanding energy consumption of sensor enabled applications on mobile phones., *in* 'Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference', Vol. 2009, pp. 6885–8.

[6] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P. [2010], Maui: Making smartphones last longer with code offload, *in* 'Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services', MobiSys '10, ACM, New York, NY, USA, pp. 49–62.

[7] Damasevicius, R., Ziberkas, G., Stuikys, V. and Toldinas, J. [2012], 'Energy Consumption of Hash Functions', *Electronics and Electrical Engineering* **18**(10), 81–84.

[8] Ellis, C. A. and Gibbs, S. J. [1989], 'Concurrency control in groupware systems', *SIGMOD Record* **18**(2), 399–407.

[9] Fangming, L., Peng, S., Hai, J., Linjie, D., Jie, Y., Di, N. and Bo, L. [2013], 'Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications', *Wireless Communications, IEEE* **20**(3), 14–22.

[10] Fangming, L., Peng, S. and Lui, J. [2015], 'AppATP: An Energy Conserving Adaptive Mobile-Cloud Transmission Protocol', *Computers, IEEE Transactions on* **64**(11), 3051–3063.

[11] Fraser, N. [2009], Differential synchronization, *in* 'Proceedings of the 9th ACM symposium on Document engineering - DocEng '09', ACM Press, New York, New York, USA, p. 13.

[12] Gao, B. and He, L. [2013], Modelling Energy-Aware Task Allocation in Mobile Workflows, *in* 'Proceedings of MobiQuitous, 2013 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services', Tokyo, Japan.

[13] Grishchenko, V. [2010], Deep hypertext with embedded revision control implemented in regular expressions, *in* 'Proceedings of the 6th International Symposium on Wikis and Open Collaboration', WikiSym '10, ACM, New York, NY, USA, pp. 3:1–3:10.

[14] Kadlček, F. and Fučík, O. [2013], Fast and Energy Efficient AdaBoost Classifier, *in* 'Proceedings of the 10th FPGAworld Conference', FPGAworld '13, ACM, New York, NY, USA, pp. 2:1–2:5.

[15] Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T. and Englund, M. [2012], CoRED: Browser-based Collaborative Real-time Editor for Java Web Applications, *in* 'Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work', CSCW '12, ACM, New York, NY, USA, pp. 1307–1316.

[16] Liu, H., Zhang, Y. and Zhou, Y. [2011], TailTheft: Leveraging the Wasted Time for Saving Energy in Cellular Communications, *in* 'Proceedings of the Sixth International Workshop on MobiArch', MobiArch '11, ACM, New York, NY, USA, pp. 31–36.

[17] Myers, E. W. [1986], 'AnO(ND) difference algorithm and its variations', *Algorithmica* **1**(1-4), 251–266.

[18] Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E. and Tarkoma, S. [2013], Carat: Collaborative energy diagnosis for mobile devices, *in* 'Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems', SenSys '13, ACM, New York, NY, USA, pp. 10:1–10:14.
URL: *http://doi.acm.org/10.1145/2517351.2517354*

[19] Oster, G., Urso, P., Molli, P. and Imine, A. [2005], Real time group editors without Operational transformation, Research Report RR-5580, INRIA.

[20] Ra, M.-R., Priyantha, B., Kansal, A. and Liu, J. [2012], Improving energy efficiency of personal sensing applications with heterogeneous multi-processors, *in* 'Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12', ACM Press, New York, New York, USA, p. 1.

[21] Rachuri, K. K., Mascolo, C., Musolesi, M. and Rentfrow, P. J. [2011], SociableSense: Exploring the Trade-offs of Adaptive Sampling and Computation Offloading for Social Sensing, *in* 'Proceedings of the 17th Annual International Conference on Mobile Computing and Networking', MobiCom '11, ACM, New York, NY, USA, pp. 73–84.

[22] Simon, J., Schmidt, P. and Pammer, V. [2014], An energy efficient implementation of differential synchronization on mobile devices, *in* 'Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services', MOBIQUITOUS '14, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, pp. 382–383.

[23] Sun, C., Jia, X., Zhang, Y., Yang, Y. and Chen, D. [1998], 'Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems', *ACM Trans. Comput.-Hum. Interact.* **5**(1), 63–108.

[24] Tan, Z., Xian, Z., Fangming, L., Hongkun, L., Qian, Y. and Guanfeng, L. [2015], eTrain: Making Wasted Energy Useful by Utilizing Heartbeats for Mobile Data Transmissions, *in* 'Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on', pp. 113–122.

[25] Weiss, S., Urso, P. and Molli, P. [2008], Logoot: a P2P collaborative editing system, Rapport de recherche RR-6713, INRIA.

[26] Weiss, S., Urso, P. and Molli, P. [2010], 'Logoot-undo: Distributed collaborative editing system on p2p networks', *IEEE Trans. Parallel Distrib. Syst.* **21**(8), 1162–1174.