ECHOES: A Fail-safe, Conflict Handling, and Scalable Data Management Mechanism for the Internet of Things

Christoph Stach and Bernhard Mitschang

Institute for Parallel and Distributed Systems, University of Stuttgart,
Universitätsstraße 38, D-70569 Stuttgart, Germany
{stachch,mitsch}@ipvs.uni-stuttgart.de

Abstract. The Internet of Things (IoT) and Smart Services are becoming increasingly popular. Such services adapt to a user's needs by using sensors to detect the current situation. Yet, an IoT service has to capture its required data by itself, even if another service has already captured it before. There is no data exchange mechanism adapted to the IoT which enables sharing of sensor data among services and across devices.
Therefore, we introduce a data management mechanism for the IoT. Due to its applied state-based synchronization protocol called ECHOES. It is fail-safe in case of connection failures, it detects and handles data conflicts, it is geared towards devices with limited resources, and it is highly scalable. We embed ECHOES into a data provisioning infrastructure, namely the Privacy Management Platform and the Secure Data Container. Evaluation results verify the practicability of our approach.

Keywords: IoT · Data Exchange · Synchronization Protocol.

1 Introduction

The *Internet of Things* (*IoT*) has long left its early stages of development behind in which only technology evangelists and early adopters used *Smart Things*[1]. Due to omnipresent Internet connectivity options and increasing bandwidth speeds, devices with a permanent Internet access grew proliferated in the early 2000s. Yet, it was not until the IoT became more and more invisible, by integrating sensors into everyday objects, that this technology found its way into limelight [7].

The Internet of Things is extremely intriguing for both consumers and service providers as the data collected by Smart Things is extremely valuable. Since these devices are usually permanently close to their users and their sensors are able to record a wide range of data, data scientists can gain profound knowledge about the users. Services can thus be tailored to individual customers. Application cases can be found in any domain, such as *Smart Homes*, *Smart Cars*, or *Smart Health*.

[1] We use the term '*Smart Thing*' for any kind of device which is able to connect to other devices in order to exchange data with each other and has the ability either to monitor its environment or to control other devices.

In: Welzer, T. et al. (Eds.) Advances in Databases and Information Systems. ADBIS 2019. Lecture Notes in Computer Science, vol 11695. Springer, Cham, pp. 373–389, 2019.

So, it is not surprising that analysts predict that by 2020 over 50 billion Smart Things will be in use and the market will be shared among many service providers [13]. Each device vendor and each service provider has its own way of storing and processing the captured data. Therefore, the effort to collect all data required for a certain service is cumbersome—different services have to collect the same data over and over again, as there is no simple data exchange mechanism among services, let alone a direct data exchange across Smart Things. Yet, the IoT can only unfold its true potential if all user data is available to all of his or her Smart Things at any time. Due to the vast amount of data, a smart synchronization mechanism is required, i. e., the volume of data being transmitted has to be as low as possible and the computational costs for Smart Things have to be minimal as these devices often have very limited resources [35].

Since none of the existing data exchange approaches for the IoT provide these characteristics, we introduce a stat**e**-based syn**ch**ronization pr**o**tocol for the Int**e**rnet of Thing**s**, called *ECHOES*, and apply it to a data exchange mechanism. To this end, we make the following three contributions:

I) We introduce an approach that enables Smart Things to exchange their data using a central synchronization server. For this purpose, our state-based synchronization protocol is used. Whenever changes to the client's data stock (i. e., adding new data sets, modifying existing data sets, or removing data sets) are done, it calls synchronization server and applies the changes to the remote data stock. Analogously, the changes are then applied to all other Smart Things of this user by the synchronization server. Due to the state-based procedure, our approach is resource-friendly in terms of run time performance and the volume of data being transmitted. ***II)*** As for Smart Things it cannot be ensured that they have a permanent connection to the server, our protocol also supports an offline mode. That is, Smart Things are able to gather data locally and synchronize with the server as soon as connectivity is back on. In this mode also, a resource-friendly proceeding is ensured. It has to be mentioned that in this stage conflicts can occur (e. g., if two offline Smart Things modify the same data set) and ECHOES is able to detect and resolve them. ***III)*** We embed ECHOES into an existing data provisioning infrastructure for Smart Things, namely the *PMP* (**P**rivacy **M**anagement **P**latform) [27] and the *SDC* (**S**ecure **D**ata **C**ontainer) [29]. The SDC is a shared data storage for IoT services and the PMP provides access to this data stock. By integrating ECHOES, we enable the synchronization of all SDCs and therefore, the PMP is able to provide all collected data of a user to any service on every Smart Thing. As the PMP provides fine-grained access control mechanisms, this data exchange is done in a privacy-aware manner.

The remainder of this paper is as follows: Section 2 introduces a real-world application case from the Smart Health domain, in which it is mandatory for Smart Things to exchange their data. From this, we derive a requirement specification in Sect. 3. We discuss related work in Sect. 4. Subsequently, we introduce the protocol of ECHOES in Sect. 5 and detail on its implementation based on the PMP and the SDC in Sect. 6. An evaluation of ECHOES is given in Sect. 7. Finally, Sect. 8 concludes this paper.

2 Application Case

In the following, we depict an IoT use case from the Smart Health domain. Smart Health applications are especially beneficial for patients suffering from chronic diseases such as diabetes. These patients have to visit their physicians regularly in order to do medical screenings and to keep track of their disease progression. Supported by Smart Things however, the patients are able to perform the required medical check-up at home by themselves. This does not only save a lot of money, but it also enables physicians to focus on emergencies [25]. *Smart Phones* are considered as particularly useful in the context of Smart Health application [11]. On the one hand, people always carry their Smart Phones with them and on the other hand, these devices are equipped with a variety of sensors which are relevant for Smart Health applications. For instance, Knöll emphasized the importance of the location where a medical reading was taken for its interpretation [14]. In addition, it is possible to connect medical devices such as glucometers or peak flow meters to Smart Phones via Bluetooth.

An example for such an application is *Candy Castle* [26]. The game is designed for children suffering from diabetes and is intended to help them coping with the continuous blood glucose monitoring. To this end, the children have to defend a virtual castle against attacks by dark forces (as a reflection of their disease). To protect the castle, the children have to build walls which hold the attackers back. They do this by doing a blood glucose reading. As the walls gradually wear down, Candy Castle motivates players to regularly check their blood sugar levels. Additionally, the virtual castle is linked to the location of the children's home and each reading is supplemented with the GPS coordinates where it was taken. Thereby, the walls can be inserted geographically correct in the game and the players are motivated to check their blood sugar levels in many different places. From a medical point of view, the children get used to the continuous blood glucose monitoring and physicians are able to identify healthy and unhealthy places in the children's environment [14]. Further sensors can be integrated to identify unhealthy factors more precisely. For instance, *Smart Bracelets* can be used to detect certain activities (e. g., administering insulin) [16], microphones can be used to determine the child's mood [18], and cameras can be used to calculate the bread units of the children's food [3]. The blood glucose measurement can also be integrated into the game automatically, e. g., via the *Apple Watch* [34].

However, such an application requires a fast and simple data exchange between the patient and his or her physician [6]. And Candy Castle is only one of many Smart Health applications. For example, a COPD application requires similar data, such as the location of a metering [31]. To avoid that every application has to acquire the same data all over again, there is a growing trend towards a *quantified self* [33]. That is, all of a user's data on a specific domain is stored in a central repository [5]. Yet, this is limited to the pre-defined types of data of the repository vendor and it is not possible to use it for a generic use case.

So, it is necessary to have a generic mechanism to easily share data between different Smart Things and applications to face the interconnection and data exchange problems and thereby exploit the full potential of IoT applications [19].

3 Requirement Specification

Although the IoT is designed as a decentralized network of independent devices, the client-server model as shown in Fig. 1 is still predominant [22]. So, a data exchange mechanism for the IoT can capitalize on the advantages of a server, namely its high availability, computing power, and scalability. None of this is provided by the Smart Things. Thus, clients should mainly focus on data gathering and processing and not the management of the data exchange. These management tasks have to meet the following requirements:

R_1 – Availability of Data: As the given application case shows, a permanent network connection must not be required to collect and process data. Users have to be able to edit data locally offline (e. g., do a blood glucose reading) and synchronize it with other devices (e. g., their physicians' devices) later when there is a network connection. This guarantees a fail safe usage of IoT applications.

R_2 – Conflict Handling: When a user edits the same data set offline on multiple devices, conflicts might occur during synchronization (e. g., a user enters contradictory readings). So, conflicts must be handled without data loss.

R_3 – Efficiency: Synchronization has to deal with limited resources of Smart Things, e. g., limited data transfer volume, limited computational power, limited memory, and limited battery. So, the amount of metadata must be minimal and the calculation of the data that has to be transferred must be simple and fast.

R_4 – Transparency: The synchronization must be transparent for the user and—as good as possible—also for application developers. That is, user interaction has to be largely avoided and the developers' effort must be minimal.

R_5 – Genericity: The mechanism must not be restricted to a certain domain or a specific data schema. The data has to be available to any application.

R_6 – Scalability: As users interact with an increasing number of Smart Things and IoT applications, many read and write data accesses must be expected. So, synchronization has to be able to cope with a large number of mobile clients.

R_7 – Security: The IoT handles a lot of private data, i. e., security is a key aspect. This applies in particular to data storage, data access, and data sharing.

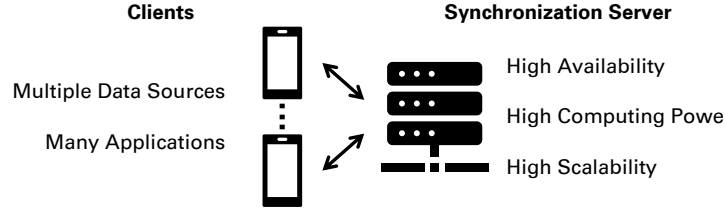


Fig. 1: Client-Server Model for the IoT.

4 Related Work

With these requirements in mind, we look at related work. We consider work from three different categories: a) data sharing approaches for Smart Things, b) Cloud-based solutions for data sharing, and c) synchronization approaches for the IoT. We focus on Android-based solutions, as this OS is becoming increasingly predominant in the IoT context, e. g., *Android Things*[2] or *emteria.OS*[3].

Smart Thing Approaches. Android's safety concept requires applications to run in sandboxes, strictly isolated from each other. A direct data exchange among applications is therefore not intended. In order to support data sharing, Android introduces so called *Content Providers*, i. e., restricted interfaces to the data sets of an application [12]. Although the interfaces are standardized, it is still cumbersome to use Content Providers. An application has to specify in the program code which Content Providers—i. e., which other applications—it needs to access. As a result, this approach largely lacks genericity. To overcome this flaw, *MetaService* [8] introduces a temporary shared object storage. In this storage, applications can deposit a single data object and another application is able to obtain it. When another object is deposited, any previously stored object is overwritten. In this respect, MetaService works similar to the *Clipboard* which is common on desktop PCs. However, MetaService is not suitable for the distribution of comprehensive data volumes. This is provided by the *SDC* [28,29]. The SDC is a shared database for Android that can be used by any application. Due to its fine-grained access control, users can specify precisely, which application may access which data set. With the *CURATOR* [30] extension, *NoSQL* databases are also supported so that, for instance, objects can be exchanged directly among applications. More information on the SDC is given in Sect. 6. However, none of these approaches supports data exchange across devices. *Mobius* [10] can be used for this purpose. It introduces a system-wide database which is synchronized with a Cloud database to share data across devices. However, Mobius uses locally virtual partitions to realize access control. An application has access to its own partition, only. That is, in order to share data with several applications the respective data has to be added to each respective partition. Although data sharing across devices is very simple, data sharing among applications is still cumbersome.

Cloud-based Approaches. There is a variety of Cloud services that enable data sharing across Smart Things. The most straightforward approach is a Cloud database in which applications can store and retrieve their data. These databases are usually associated with a specific application or class of applications and have a fixed data schema. For instance, there is the *Glucose Web Portal* [15], which can be used by applications related to diabetes. In addition to the storage of data, the Glucose Web Portal also provides some health services, e. g., the analysis of diabetes data. That is, an application has to collect health data and send it to the service and then the not only the data itself but also the analysis results are

[2] see https://developer.android.com/things/
[3] see https://emteria.com/

available to any other application. Similar services are available for other diseases such as COPD as well [32]. A more generic approach is the *HealthVault* [5]. A user has to create a HealthVault profile. Then s/he can store and link any kind of health data captured by any Smart Thing in his or her profile. This data is available for any application for its analysis and presentation. However, these approaches do not support user-defined types of data, an application automatically has access to all data, and since the data is only stored in the Cloud, it is not available if the Internet connection is interrupted. Many database vendors also provide a mobile version of their databases. These versions often support synchronization with a Cloud-based back-end. *Couchbase Mobile*[4] is such a mobile version. However, its synchronization is designed to ensuring that the data sets of a particular application are kept up to date on all of a user's Smart Things. Data sharing among different applications is not supported. Also, the synchronization only operates with the mobile client from Couchbase—other databases are not supported. *Resilio Sync*[5] improves the availability of such a synchronization by adopting a *P2P* approach instead of a central database [23]. Yet, this enables only the exchange of files and not of single data sets and it has fairness issues, which cause significant slowdowns [20].

Synchronization Approaches. *Syxaw* [17] brings the two aforementioned categories together by introducing a middleware for the synchronization of structured data. It enables multiple users to edit documents and folders collaboratively, and Syxaw takes care of merging the changes. However, since it operates on files, the computation of changes is expensive and locks on at file level are very restrictive. For a use case as presented in Sect. 2, a fine-grained synchronization of data sets is much better suited. This is achieved by *SAMD* [9]. In order to reduce the computational effort for the mobile clients, all expensive operations are carried out server-sided. This includes a multi-layered calculation of hash values for the managed data sets. Thereby it is sufficient to exchange comparatively small hash values for most of the synchronization process instead of the actual data. *SWAMD* [2] follows a quite similar approach, but its focus is on wireless networks, which is common in an IoT environment. Yet, both approaches are designed for a deployment scenario in which synchronization takes place infrequently. A continuous synchronization of data requires a permanent recalculation and exchange of the hash values, which causes high costs. Contrary to this, *MRDMS* [24] represents a timestamp-based synchronization approach. The timestamps enable MRDMS to reflect the temporal correlation of changes. In this way, the required data transfer volume can be further reduced compared to SAMD. However, since less data is used for synchronization, conflicts often cannot be resolved. Furthermore, *lost updates* cannot be prevented. By incorporating *snapshots* into such approaches, automatic conflict resolution can be improved [21]. Yet, this increases computational effort and data volumes.

As none of these approaches supports a use case as given in Sect. 2 as well as the requirements from Sect. 3, we introduce a solution in the following.

[4] see https://www.couchbase.com/products/mobile
[5] see https://www.resilio.com

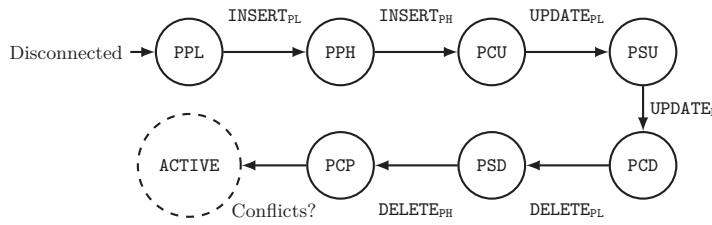
5 The ECHOES Protocol

In ECHOES, we do not pursue a P2P approach as it is not guaranteed that all Smart Things are permanently available and interconnected. A central, permanently available server component therefore ensures the fastest and most reliable data distribution. Moreover, computation-intensive tasks can be shifted to the server in order to reduce computational effort for the Smart Things. A one-way *push* or *pull* approach is inefficient as data changes can occur on both the server and the clients. Therefore, we apply a two-way state-based approach in our synchronization protocol. The synchronization steps can be simplified, since it is possible to decide based on the respective state which actions are necessary. Furthermore, we introduce version numbers for conflict resolution.

Offline Mode. For the initial synchronization or after a connection failure, the client has to process seven states sequentially (see Fig. 2). This is due to the fact that during the offline period various changes may have been made to the databases of both the client and the server.

First, it is necessary to check whether new data sets have been added. In the *primary pull* (`PPL`) the client sends all IDs of its data sets to the server and the server calculates the delta to its central database. As a response to the `PPL`, the server sends all data sets that are not available on the client. The client sets the status of these data sets to `RELEASED`. The client then performs a *primary push* (`PPH`) by sending all data sets that were added in the offline stage—i. e., data sets with the status `NEW`—to the server. When the server acknowledges receipt, their statuses are set to `RELEASED`.

ECHOES then handles edited data sets. In the *primary client update* (`PCU`) the client sends all version numbers of data sets with status `RELEASED` that were not handled by the previous steps to the server. The version number of a data set is incremented when a `RELEASED` data set is edited. The server sends back updates for all data sets for which a newer version exists. In the *primary server update* (`PSU`), the client sends then all data sets with the status `MODIFIED`—i. e., data sets that were edited during offline stage—to the server. The server checks based on the version number whether it can apply the update or whether there



`PPL`: **P**rimary **P**ull `PPH`: **P**rimary **P**ush `PCU`: **P**rimary **C**lient **U**pdate `PSU`: **P**rimary **S**erver **U**pdate
`PCD`: **P**rimary **C**lient **D**elete `PSD`: **P**rimary **S**erver **D**elete `PCP`: **P**rimary **C**onflict **P**ull

Fig. 2: ECHOES Offline Synchronization Process (applies to server and client).

is a conflict with a change made by another client. Accordingly, the status on the client is set to `RELEASED` or the conflict is logged.

From the previous steps, the server already knows all the unmodified data sets on the client (status `RELEASED`). If these sets do not exist on the server any more, they must be deleted on the client as part of the *primary client delete* (`PCD`). Accordingly, the *primary server delete* (`PSD`) synchronizes local deletions. To this end, deletions performed during offline mode are not applied to the data stock immediately, but the status of the affected data sets is set to `DELETED` and the version numbers are incremented. During `PSD`, the server checks whether the data can be deleted or whether there is a conflict and gives feedback to the client.

Finally, ECHOES deals with conflict resolution. For all data sets flagged as conflicting, the client receives the versions available on the server. As these conflicts cannot be resolved automatically, the *primary conflict pull* (`PCP`) requires user interaction. The user has to decide which version is the valid one. The version number of this data record is then adjusted. The new version number is the maximum of the two former version numbers plus 1.

After these seven steps, the online mode is activated. Nota bene, authentication and authorization of the Smart Things towards the synchronization server are not part of the ECHOES protocol. This has to be done in a preliminary step. ECHOES handles synchronization, only. Yet, we tackle both of these issues in our implementation (see Sect. 6).

Online Mode. In online mode (see Fig. 3), client and server mutually send acknowledgement messages periodically as a heartbeat message. In this process, both change their state from `ACTIVE` to `STANDBY` and vice versa. That way, no permanent (energy-consuming) connection is necessary.

Each party (i. e., client and server) can continue to work and process data locally, regardless of its current state. To this end, each party adds corresponding tasks to a local queue. This queue is processed as soon as the respective party is active. Each of these tasks refers to a single data set, only. As a result, the processing is significantly less computationally expensive than the synchronization in offline mode and only a single state has to be traversed per task.

A reasonable tradeoff must be achieved in this respect. Long `ACTIVE-STANDBY` cycles cause less communication overhead (for passing the activity token, i. e., the heartbeat message), but more local changes—and therefore potential conflicts—might occur per cycle and it takes more time until the changes are applied to all devices. Short cycles cause increased communication overhead, even if there have not been any changes during the `STANDBY` phase.

Immediate pull (`IPL`) and *immediate push* (`IPH`) are the counterparts in the online mode to the `PPL` and `PPH` state in the offline mode. Conflicts do not have to be considered in these states. The IDs used by ECHOES contain a reference to the Smart Thing that generates the data. This prevents conflicts if new data is simply added.

When data sets are edited, an *immediate update pull* (`IUPL`) or respectively an *immediate update push* (`IUPH`) is triggered. First, the server checks based on the version number whether the changes can be applied immediately to all

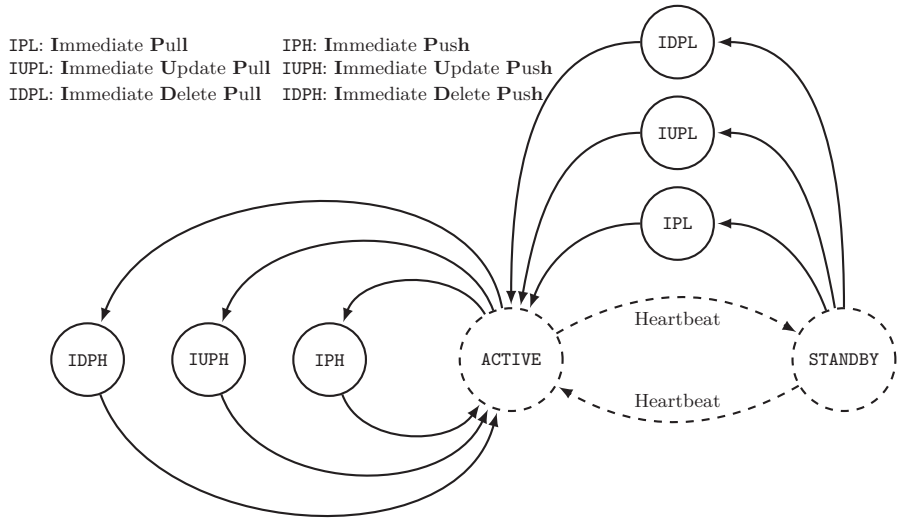


Fig. 3: ECHOES Online Synchronization Process (applies to server and client).

clients. If this is the case, the status is set to `RELEASED`. If there is a conflict, the server attempts to resolve it by merging the changes. If that is successful, the new version of the data set is distributed to all clients. If the conflict cannot be resolved automatically, the client that submitted the change is notified and the user must resolve the conflict manually. To prevent the synchronization from being blocked by this user interaction, the data set is marked as a conflict and the update task is added to the client's queue again. Once the conflict is resolved, the conflict flag as well as the update task are removed, and the changes are synchronized by the server on all clients.

Finally, *immediate delete pull* (`IDPL`) and *immediate delete push* (`IDPH`) deal with the synchronization of delete operations. The client (or analogously the server) receives the ID and the version number of the data set in question. If the version number is equal to or higher than the one of the local instance, the data set is immediately deleted. Otherwise the local version was edited, and the corresponding synchronization has not been carried out yet—i. e., the `IUPH` task is still in the client's queue. In such a case, the deletion operation is refused, and the user is informed about the conflict. If the conflict occurs on the server, the resolution takes place on the client that caused the conflict. A dedicated conflict state is not required in online mode, since conflicts are resolved immediately when they occur.

Although ECHOES enables synchronization across devices, there is still a lack of a data exchange mechanism among applications. Therefore, an implementation is presented in the following, in which ECHOES operates as a background service that is available to any application.

6 Implementation of ECHOES

The PMP [27] is a privacy system for Smart Things. To this end, it isolates applications form data sources and controls any access to the data via its fine-grained permissions. In other words, the PMP can be seen as a middleware that operates as an information broker. The PMP's key feature is that it is extendable. New data sources can be added at any time as so-called *Resources*. Subsequently, applications can access these data sources via the PMP.

The SDC [29], a database system based on *SQLite*, is such a Resource. It offers security features, e. g., the stored data is encrypted and it provides a tuple-based access control. In addition, it has a customizable schema to be compatible to any application and stored data can be partitioned to increase performance.

As the SDC is a PMP Resource, it is available to any applications. Thus, data can be exchanged among applications via the fine-grained access control. By integrating ECHOES into the SDC, an exchange across devices can be realized.

Figure 4 shows the data access and synchronization process of an SDC-based ECHOES implementation. Initially, an application requests access to the local SDC instance from the PMP (1). Then, the PMP checks whether the application has the required permissions. If the application is authorized to use the SDC, the PMP enables access. The application can use the SDC like an internal database, i. e., it can query, insert, update, or delete data (2). If data from another application is affected, the SDC checks the required access rights.

Due to the integration of ECHOES, the SDC detects changes to the data stock and synchronizes it (3). Depending on whether the SDC is currently in offline or online mode, this involves different steps. If it is in offline mode, it must establish a connection to the synchronization server and perform a complete synchronization (see Fig. 2). Otherwise, the respective type of alteration can be

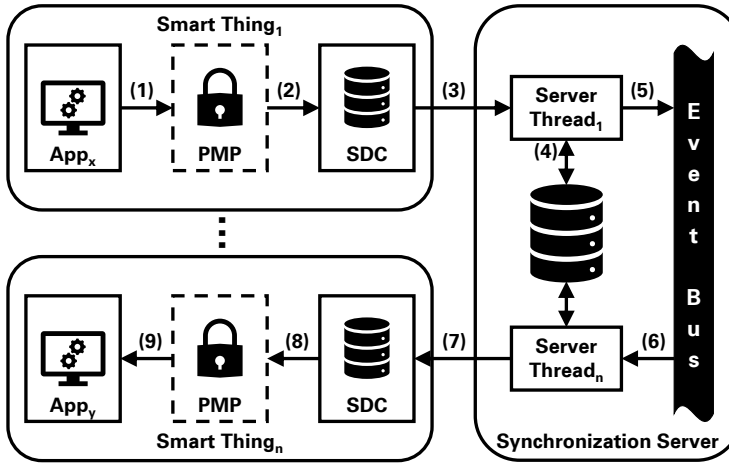


Fig. 4: Data Access and Synchronization Process.

ECHOES_DATA_SETS					
ID	MODE	VERSION	TIMESTAMP	SHARE	PAYLOAD
INT				FK	DATA

Fig. 5: Relational Schema for the Mobile Database.

submitted to the server (see Fig. 3). In this process, the changes are applied to the central database (4). The synchronization server then creates an event to notify all other SDC instances via its event bus (5). These notifications are transformed into tasks and added to the threads of the respective Smart Things (6). Finally, the necessary changes are applied to their local SDC instances (7). The data from Smart Thing_1 is then available to applications on Smart Thing_n (8). Again, the PMP handles access control (9).

For the integration of ECHOES into the SDC, its data schema has to be adapted, as additional metadata is required for synchronization. The extended relational schema is shown in Fig. 5. First, a new *ID* is added for each data set. This ID contains references to the Smart Thing that created or edited the data set most recently. In addition, the *mode* of each data set has to be logged. In addition to the four modes (`NEW`, `RELEASED`, `MODIFIED`, and `DELETED`) which are required for synchronization (see Sect. 5), there is a fifth mode `OFFLINE`. Data sets flagged with this mode are excluded from synchronization. If a conflict cannot be resolved automatically, this is also indicated in the mode entry of the corresponding data sets. The SDC then informs the user about the conflict and s/he can decide which version should be valid. The *VERSION* of the data sets is required by ECHOES to decide which version is valid. The *TIMESTAMP* entry is not necessarily required for the synchronization as the version already represents the chronological order in which the data was edited. Yet, the timestamps help users to resolve conflicts as they are able to track exactly when which data set was edited. *SHARE* is provided by the SDC. It is a foreign key to the maintenance tables of the SDC. These tables have to be synchronized as well in order to enable access control on all devices. Finally, the actual *PAYLOAD* is stored as well. An individual data schema can be specified, just like in the native SDC.

The actual data transfer is realized as a flattened stream of characters. A composer in the SDC converts the database entries into a sequence of key-value pairs and a parser processes such a sequence and inserts the contained values into the SDC. That way, the amount of data that needs to be transferred is minimized as almost solely payload data is transferred. On the synchronization server, there are corresponding counterparts according to the specifications in Sect. 5.

7 Evaluation

To evaluate the performance of our ECHOES prototype, we describe the evaluation setup, present the evaluation results, and discuss whether ECHOES fulfills all requirements towards a data exchange mechanism for the IoT.

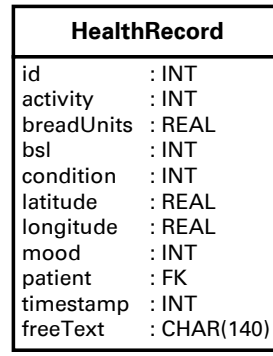


Fig. 6: Candy Castle Data Model Used for Evaluation.

Evaluation Setup. For the evaluation, we draw on Candy Castle [26]. This application is executed on two Smart Phones and the captured data is synchronized on both devices. For this purpose, we extended the SDC-based data management of Candy Castle by ECHOES—the applied data model of the payload is shown in Fig. 6. In addition, we set up an ECHOES synchronization server. To get a better understanding of how different hardware configurations and different Android versions affect the performance of ECHOES, we perform our measurements on two different types of Smart Phones: on the one hand the *LG Nexus 5X* (S_1) with a current Android version and on the other hand the *Huawei Honor 6 Plus* (S_2) with more memory and more CPU cores but a lower clock speed. Both are intentionally lower middle-class models, since their hardware setup is similar to those of other popular Smart Things, such as the *Raspberry Pi*. *MariaDB* is used on the server. MariaDB is a highly powerful and scalable database with strong similarities to *MySQL* [1,4]. A detailed evaluation setup is given in Table 1.

For the evaluation, we examine four different scenarios for both, the offline mode as well the online mode: a) Initially, both Smart Phones are disconnected and an ascending number of data sets (from 100 up to 6,400) is randomly generated on one Smart Phone. Then, both devices are connected. b) Subsequently,

Table 1: Evaluation Setup.

	Smart Thing S_1	**Smart Thing** S_2	**Server**
OS	Android 8.1.0	Android 5.1.1	Debian 9.4
CPU	Snapdragon 808	Kirin 925	$8 * 3.6$ GHz
RAM	2 GB	3 GB	8 GB
Connection	50 Mbit/s	50 Mbit/s	100 Mbit/s
Database	SQLite	SQLite	MariaDB

both Smart Phones are disconnected again. Then, on one device 50 % of the data sets are edited and synchronization is started. c) Next, 50 % of the data is edited on both devices in offline mode, i. e., ECHOES must resolve conflicts. d) Finally, 50 % of the data sets are deleted on one device and synchronization is started.

In each scenario, we take the time until synchronization is completed. These scenarios are repeated with permanently enabled connectivity to evaluate online mode. In this case, the duration of the synchronization of each individual operation is measured. Each test is performed with a pair of Smart Thing S_1 and S_2. After each run, all databases are reset to avoid side effects caused by warm caches. Each test is carried out for 10 times and the average processing time is considered.

Evaluation Results. All evaluation results for ECHOES's offline mode are shown in Fig. 7. Figure 7a shows the time until newly added data sets are available on all devices (`PPL` & `PPH`). The processing time increases nearly linear to the number of data sets. On average, the synchronization of a single newly added data set takes about 84 ms on a pair of S_1 and about 99 ms on a pair of S_2. It is striking that ECHOES is performing very well on the weaker hardware. A considerably more decisive factor is the OS version. These findings are also reflected by the three other scenarios.

The processing time when changes are made to 50 % of the data sets (`PCU` & `PSU`) is shown in Fig. 7b. This processing time also increases linearly, but it is significantly higher than in the previous scenario. Although only half of the data sets have been changed, the other half must also be cross-checked with the server. Nevertheless, a processing time of 180 ms or 215 ms per edited data set is still reasonable.

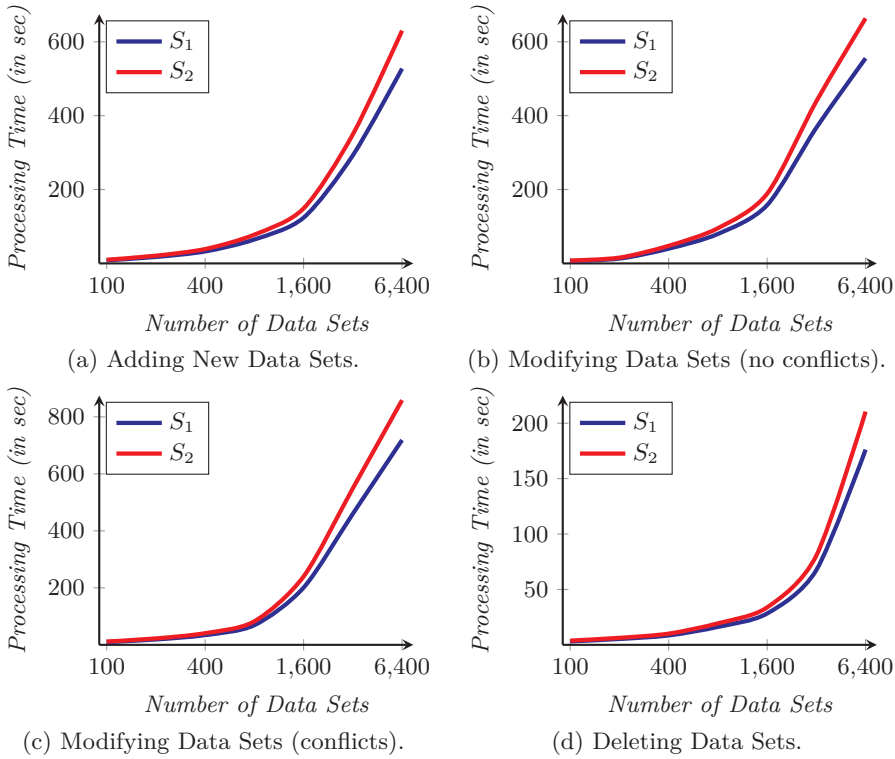
The effect of conflict resolving (`PCP`) on the processing time is shown in Fig. 7c. This conflict resolving increases the costs caused by ECHOES (213 ms or 254 ms per edited data set). Yet, conflicts are unlikely in our application case.

The deletion of data sets (`PCU` & `PSU`) is very fast (24 ms or 28 ms per data set, see Fig. 7d).

In the online mode, changes (regardless whether it is add, modify, or delete) are available on all devices after about 350 ms. The detailed costs are stated in Table 2. As shown in Sect. 5, conflicts need not to be considered explicitly in online mode since they are handled by the three listed operations already. The online synchronization takes longer than the synchronization of a single data set in offline mode, as the communication overhead required to initiate the data exchange is generated only once for the total bulk of transferred data. These costs

Table 2: Processing Time of ECHOES's Online Mode.

	Add	**Modify**	**Delete**
Smart Thing S_1	365 ms	368 ms	309 ms
Smart Thing S_2	398 ms	402 ms	337 ms



(a) Adding New Data Sets.

(b) Modifying Data Sets (no conflicts).

(c) Modifying Data Sets (conflicts).

(d) Deleting Data Sets.

Fig. 7: Overall Processing Time of ECHOES's Offline Mode.

are thus allocated proportionally in offline mode among all data sets contained in the bulk.

Discussion. As ECHOES has an online and an offline mode, it ensures **availability** even in case of connection failures. This enables users to continue working and ECHOES takes care of the synchronization as soon as the connection is reestablished ($\mathbf{R_1}$). *Conflict handling* is ensured, as ECHOES resolves conflicts automatically due to its `PCP` state ($\mathbf{R_2}$). Evaluation results prove ECHOES **efficiency**. Not only does it cope with limited resources, but also the required metadata is minimal. Based on the memory consumption of an SQL database, the payload in our application case requires 192 bytes per data set, while the metadata occupies only 20 bytes. That is almost a ratio of 10 to 1. Obviously, this is case-specific ($\mathbf{R_3}$). From an application's point of view, **transparency** is achieved. An application interacts with the SDC as if it is a local database ($\mathbf{R_4}$). Also, **genericity** is ensured at the data schema of an SDC instance can be customized ($\mathbf{R_5}$). The server-side **scalability** is ensured by the use of MariaDB and as the SDC can be partitioned, the scalability can be further improved ($\mathbf{R_6}$). Finally, the PMP and the SDC ensures privacy and **security** on the mobile

clients ($\mathbf{R_7}$). Therefore, ECHOES meets all requirements towards a data exchange mechanism for the IoT.

8 Conclusion

The IoT is becoming increasingly popular. A growing number of applications are emerging in various domains such as Smart Homes, Smart Cars, or Smart Health. These IoT applications require a mechanism to share data. However, current data sharing approaches do not fulfills all requirements towards such a mechanism.

Therefore, we introduce ECHOES, a state-based synchronization protocol for the IoT. It provides four key features: 1) It supports an online and offline mode to deal with connection failures. 2) It deals with conflicts when several parties edit the same data set. 3) It can be executed on limited resources. 4) It operates with any given data schema. We implement this protocol in a data provisioning infrastructure for Smart Things (PMP & SDC). Thus, our prototype has three further key features: 5) The SDC behaves like a local database. 6) The back-end is highly scalable due to MariaDB. 7) The PMP and the SDC provide a wide range of data security features. Evaluation results are very promising as changes are available on all clients in less than 0.4 seconds.

Acknowledgment

We thank the BW-Stiftung for financing the PATRON research project and the DFG for funding the SitOPT research project.

References

1. Aditya, B., Juhana, T.: A high availability (HA) MariaDB Galera Cluster across data center with optimized WRR scheduling algorithm of LVS – TUN. In: TSSA '15 (2015). https://doi.org/10.1109/TSSA.2015.7440452
2. Alhaj, T.A., Taha, M.M., Alim, F.M.: Synchronization wireless algorithm based on message digest (SWAMD) for mobile device database. In: ICCEEE '13 (2013). https://doi.org/10.1109/ICCEEE.2013.6633944
3. Almaghrabi, R., Villalobos, G., Pouladzadeh, P., Shirmohammadi, S.: A novel method for measuring nutrition intake based on food image. In: I2MTC '12 (2012). https://doi.org/10.1109/I2MTC.2012.6229581
4. Bartholomew, D.: MariaDB vs. MySQL. White paper, Monty Program Ab (2012)
5. Bhandari, V.: Enabling Programmable Self with HealthVault: An Accessible Personal Health Record. O'Reilly Media Inc. (2012)
6. Chan, M., Estève, D., Fourniols, J.Y., Escriba, C., Campo, E.: Smart Wearable Systems: Current Status and Future Challenges. Artificial Intelligence in Medicine **56**(3), 137–156 (2012). https://doi.org/10.1016/j.artmed.2012.09.003
7. Chase, J.: The Evolution of the Internet of Things. White paper, Texas Instruments (2013)

8. Choe, H., Baek, J., Jeong, H., Park, S.: MetaService: An Object Transfer Platform Between Android Applications. In: RACS '11 (2011). https://doi.org/10.1145/2103380.2103391
9. Choi, M.Y., Cho, E.A.C., Park, D.H., Moon, C.J., Baik, D.K.: A Database Synchronization Algorithm for Mobile Devices. IEEE Transactions on Consumer Electronics **56**(2), 392–398 (2010). https://doi.org/10.1109/TCE.2010.5505945
10. Chun, B.G., Curino, C., Sears, R., Shraer, A., Madden, S., Ramakrishnan, R.: Mobius: Unified Messaging and Data Serving for Mobile Apps. In: MobiSys '12 (2012). https://doi.org/10.1145/2307636.2307650
11. Dayer, L., Heldenbrand, S., Anderson, P., Gubbins, P.O., Martin, B.C.: Smartphone medication adherence apps: Potential benefits to patients and providers. Journal of the American Pharmacists Association **53**(2), 172–181 (2013). https://doi.org/10.1331/JAPhA.2013.12202
12. Enck, W., Ongtang, M., McDaniel, P.: Understanding Android Security. IEEE Security Privacy **7**(1), 50–57 (2009). https://doi.org/10.1109/MSP.2009.26
13. Hung, M. (ed.): Leading the IoT: Gartner Insights on How to Lead in a Connected World. Gartner (2017)
14. Knöll, M.: Diabetes City: How Urban Game Design Strategies Can Help Diabetics. In: eHealth '08 (2009). https://doi.org/10.1007/978-3-642-00413-1_28
15. Koutny, T., Krcma, M., Kohout, J., Jezek, P., Varnuskova, J., Vcelak, P., Strnadek, J.: On-line Blood Glucose Level Calculation. Procedia Computer Science **98**(C), 228–235 (2016). https://doi.org/10.1016/j.procs.2016.09.037
16. Kwapisz, J.R., Weiss, G.M., Moore, S.A.: Activity Recognition Using Cell Phone Accelerometers. ACM SIGKDD Explorations Newsletter **12**(2), 74–82 (2010). https://doi.org/10.1145/1964897.1964918
17. Lindholm, T., Kangasharju, J., Tarkoma, S.: Syxaw: Data Synchronization Middleware for the Mobile Web. Mobile Networks and Applications **14**(5), 661–676 (2009). https://doi.org/10.1007/s11036-008-0146-1
18. Mehta, D.D., Zañartu, M., Feng, S.W., Cheyne II, H.A., Hillman, R.E.: Mobile Voice Health Monitoring Using a Wearable Accelerometer Sensor and a Smartphone Platform. IEEE Transactions on Biomedical Engineering **59**(11), 3090–3096 (2012). https://doi.org/10.1109/TBME.2012.2207896
19. Murnane, E.L., Huffaker, D., Kossinets, G.: Mobile Health Apps: Adoption, Adherence, and Abandonment. In: UbiComp/ISWC '15 Adjunct (2015). https://doi.org/10.1145/2800835.2800943
20. Peng, Z., Pallelra, R.R., Wang, H.: On the measurement of P2P file synchronization: Resilio Sync as a case study. In: IWQoS '17 (2017). https://doi.org/10.1109/IWQoS.2017.7969177
21. Phatak, S.H., Nath, B.: Transaction-centric Reconciliation in Disconnected Client-server Databases. Mobile Networks and Applications **9**(5), 459–471 (2004). https://doi.org/10.1023/B:MONE.0000034700.03069.48
22. Ren, J., Guo, H., Xu, C., Zhang, Y.: Serving at the Edge: A Scalable IoT Architecture Based on Transparent Computing. IEEE Network **31**(5), 96–105 (2017). https://doi.org/10.1109/MNET.2017.1700030
23. Scanlon, M., Farina, J., Kechadi, M.T.: Network Investigation Methodology for BitTorrent Sync. Computers and Security **54**(C), 27–43 (2015). https://doi.org/10.1016/j.cose.2015.05.003
24. Sethia, D., Mehta, S., Chowdhary, A., Bhatt, K., Bhatnagar, S.: MRDMS-mobile replicated database management synchronization. In: SPIN '14 (2014). https://doi.org/10.1109/SPIN.2014.6777029

25. Silva, B.M.C., Rodrigues, J.J., de la Torre Díez, I., López-Coronado, M., Saleem, K.: Mobile-health: A Review of Current State in 2015. Journal of Biomedical Informatics **56**(August), 265–272 (2015). https://doi.org/10.1016/j.jbi.2015.06.003
26. Stach, C.: Secure Candy Castle — A Prototype for Privacy-Aware mHealth Apps. In: MDM'16 (2016). https://doi.org/10.1109/MDM.2016.64
27. Stach, C., Mitschang, B.: Privacy Management for Mobile Platforms – A Review of Concepts and Approaches. In: MDM'13 (2013). https://doi.org/10.1109/MDM.2013.45
28. Stach, C., Mitschang, B.: Der Secure Data Container (SDC) – Sicheres Datenmanagement für mobile Anwendungen. Datenbank-Spektrum **15**(2), 109–118 (2015). https://doi.org/10.1007/s13222-015-0189-y
29. Stach, C., Mitschang, B.: The Secure Data Container: An Approach to Harmonize Data Sharing with Information Security. In: MDM'16 (2016). https://doi.org/10.1109/MDM.2016.50
30. Stach, C., Mitschang, B.: CURATOR—A Secure Shared Object Store: Design, Implementation, and Evaluation of a Manageable, Secure, and Performant Data Exchange Mechanism for Smart Devices. In: SAC'18 (2018). https://doi.org/10.1145/3167132.3167190
31. Stach, C., et al.: The Privacy Management Platform: An Enabler for Device Interoperability and Information Security in mHealth Applications. In: HEALTHINF'18 (2018). https://doi.org/10.5220/0006537300270038
32. Steimle, F., Wieland, M., Mitschang, B., Wagner, S., Leymann, F.: Extended provisioning, security and analysis techniques for the ECHO health data management system. Computing **99**(2), 183–201 (2017). https://doi.org/10.1007/s00607-016-0523-8
33. Swan, M.: Sensor Mania! The Internet of Things, Wearable Computing, Objective Metrics, and the Quantified Self 2.0. Journal of Sensor and Actuator Networks **1**(3), 217–253 (2012). https://doi.org/10.3390/jsan1030217
34. Wakabayashi, D.: Freed From the iPhone, the Apple Watch Finds a Medical Purpose. Report, The New York Times (2017)
35. Walker, M.: Hype Cycle for Emerging Technologies, 2018. Market analysis, Gartner (2018)

All links were last followed on August 13, 2019.