STCP: Simple Transaction Commit Protocol for Wireless Sensor Networks

Youqing Guan*, Ke Zhang† and Yanran Guan‡
*School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
Email: guanyouq@njupt.edu.cn
†ZTE Corporation, Nanjing 210012, China
Email: zhang.ke106@zte.com.cn
‡School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada
Email: yanranguan@cmail.carleton.ca

***Abstract*—To effectively ensure data consistency is a challenge in today's wireless sensor network (WSN) applications. This paper, based on the simplification of traditional transaction processing from database management system (DBMS), puts forward the concepts of update transaction and query transaction within the context of WSNs and proposes a novel transaction processing protocol, the simple transaction commit protocol (STCP), for WSN systems. In STCP, the base station, as coordinator, is responsible for initializing a transaction and broadcasting the transaction to participant nodes. The participants, on the other hand, commit the transaction according to a timer, which enables the participants to send acknowledgment or conflict messages to the base station at a regular time interval. The transactions suspended by conflict are awakened through a triggering mechanism. Compared to the traditional protocols of distributed systems, STCP, which adapts better to the characteristics and requirements of WSNs, can effectively and efficiently ensure data consistency in WSN systems.**

***Index Terms*—Data consistency, STCP, transaction processing, WSN.**

I. INTRODUCTION

Wireless sensor network (WSN) is a distributed system consisting of one base station and multiple participant nodes that have limited energy and computational resources. It can also be viewed as a distributed database that collects, stores, and indexes the sensor data to respond to the queries received from external users/applications as well as internal system entities. Nowadays, data management in WSNs has become an area that draws increasing attention. Many new technologies have been developed for WSN data management, e.g., continuous queries [1] and data stream processing [2], approximate query answering [3], in-network data aggregation [4], and data fusion [5], as well as query processing systems such as Cougar [6], TinyDB [7], Corona [8], and RTQPS [9].

Meanwhile, research regarding WSN transaction management is still rarely seen. There are two main reasons for it: firstly, the sensor nodes with limited resources are difficult to be made compatible with a complex protocol that ensures reliable data processing; secondly, the traditional WSN systems are mainly designed for read-only data queries that do not take data update into consideration, so that there is no need for an explicit control on data consistency. However, with the growing diversity and complexity of WSN applications, the precise control of ensuring the consistency of the sensor data becomes particularly important on many occasions. For instance, in the process of updating the sensor sampling rate for a certain factory workshop, an unsuccessful update for a few sensors will bring about inconsistency in sampling frequency in the subsequent data sampling, which will lead further to statistical errors in the results being sent to the base station. In another case, assuming that the WSN is doing continuous queries to calculate the average temperature inside a warehouse when an update operation is called that modifies the unit of temperature from Fahrenheit to Celsius, the results of the continuous queries will then be incorrect due to the changes in the unit of measurement.

The existing atomic commit protocols of distributed systems, such as the two-phase commit (2PC) protocol, cannot address these issues perfectly. Considering the limited node energy and the usually occurring node failure or communication failure in WSNs, such complicated protocols can easily cause system blocking or data inconsistency. Therefore, some transactional solutions with WSN characteristics have been proposed. The earliest idea of incorporating the transactional properties of atomicity, consistency, isolation, and durability (ACID) from database management system (DBMS) into WSN management was proposed by Gürgen et al. [10]. Reinke et al. [11] suggest ensuring the atomic property of service migration operations in service-oriented architecture (SOA) based WSN systems [12], which helps to extend the life cycle of WSNs. Based on the same idea, the cross-layer commit protocol (CLCP) [13] and the 2PC with caching (2PCwC) protocol [14], [15] were developed. Besides, with the development of blockchain technology, some recent studies proposed from the perspective of blockchain transactions [16], [17] can also be applied in WSNs.

Based on simplified DBMS transactions, we propose a completely defined transaction processing protocol for WSNs, the simple transaction commit protocol (STCP), that works for all transactional situations in WSN systems. Compared to the aforementioned transactional models, STCP has the following

This work was supported by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant No. 05KJD520146).

characteristics:

- The transactions are committed periodically based on a timer, making it possible to coordinate transactions and to have time to handle the cancel operations.
- Each participant responds to the coordinator at a regular time interval defined by the timer, reducing not only conflicts within the network but also the retransmission of messages.
- Transactional conflicts are handled in an optimistic locking manner, and a triggering mechanism is used for automatically waking up transactions that are suspended by conflicts.

II. WSN Transaction Processing Model

In this section, we introduce the transaction processing model of STCP, including the definition of WSN transaction, the atomic transaction processing algorithm, and the concurrency controller. All the SQL statements that occur in this section are written in TinyDB's SQL-like language.

A. WSN Transaction

In STCP, based on the simplification of traditional DBMS transactions, we define the update and query operations in WSN systems as WSN transactions. Detailed definitions are given below.

Definition 1. *An update transaction in WSNs, or a WSN update, consists in modifying the metadata information (such as identifier, location, sampling rate, and sampling unit) of sensor nodes.*

Owing to the distributed architecture of WSNs, data inconsistency is likely to happen when the base station modifies the information of participant nodes. We propose the concept of WSN update. For example, a WSN update can be written in the following SQL statement that raises the sampling rate by two times of the temperature sensors located at department A.

```
UPDATE sensor_attr
    SET sampling_rate = sampling_rate * 2
WHERE location = 'A' AND type = 'temperature'
```

Definition 2. *A query transaction in WSNs, or a WSN query, refers to the set of continuous queries for the streaming data from sensor nodes.*

In order to facilitate the handling of concurrency conflicts between updates and queries in WSN systems, we define WSN query as the set of statistically continuous queries for the data on sensor nodes over a period of time. For example, the following SQL code describes a WSN query that queries every 20 s in 5 min the average temperature from the temperature sensors of department A.

```
SELECT avg(temp)
    FROM sensors
WHERE location = 'A' PERIOD 20s FOR 300s
```

WSN transactions are derived from DBMS transactions, having the commit and rollback operations as well. Yet as a simplified traditional transaction, a WSN update modifies only one type of metadata information each time, and a WSN query is a consolidation of continuous traditional one-time queries.

B. Atomic Transaction Processing

As mentioned in Section I, the traditional atomic commit protocols, e.g., 2PC, are not suitable for WSN systems. The variants of traditional protocols, e.g., CLCP and 2PCwC, work under the circumstances where only a part of the participants is included, such as service migration. Therefore, we propose a new atomic commit protocol based on WSN transactions. The formal description of a WSN transaction is given in Definition 3 below.

Definition 3. *A WSN transaction is denoted by a three-tuple, written as* `WSNT(TID,TCTX,P)`*, where* `TID` *refers to the numeric identifier of the transaction,* `TCTX` *refers to the context of the transaction, and* `P` *refers to the collection of all participants.*

In Definition 3, `TID` is a randomly generated integer that identifies the transaction initiated by the base station, such as 35306. `P` is the set of all participant nodes relevant to the WSN transaction, such as $\{node_1, node_2, node_3, \dots\}$.

To describe the transaction context `TCTX` in Definition 3, we use another three-tuple, written as `TCTX(TType,TOperation,TTimeInterval)`, where `TType` refers to the type of transaction, i.e., the WSN update `WSN_UPDATE` or the WSN query `WSN_QUERY`, `TOperation` describes the detailed operation that the transaction carries out, and `TTimeInterval` denotes the time interval in which feedback messages should be sent from the participant nodes to the base station.

The `TCTX` of a WSN update is written in SQL below.

```
TCTX(
    /* transaction type */
    WSN_UPDATE,
    /* transaction operation */
    UPDATE sensor_attr
        SET sampling_rate = sampling_rate * 2
    WHERE location = 'A' AND type = 'temperature',
    /* time interval (in milliseconds) */
    1650
)
```

The atomic commit protocol of STCP comprises the coordinator part and the participant part. In our design, the state transitions on the coordinator side and the participant side are shown in Fig. 1.

The base station, being the coordinator, as shown in Fig. 1a, starts a transaction from the initial state, where the transaction is initialized and the timer of coordinator `TC` is set up. The coordinator then broadcasts the `TCTX` to all relevant participants, triggering the transition to the collecting state, and starts to collect the messages which are sent by the participants through a retransmission mechanism, where the participants have to resend the message if the coordinator fails to receive it. At the collecting state, as soon as the conflict message `CONFLICT` is received, the coordinator will abort the transaction: stop `TC`, broadcast the cancellation message `CANCEL`, and make a state transition to canceled. If the coordinator receives `ACK`

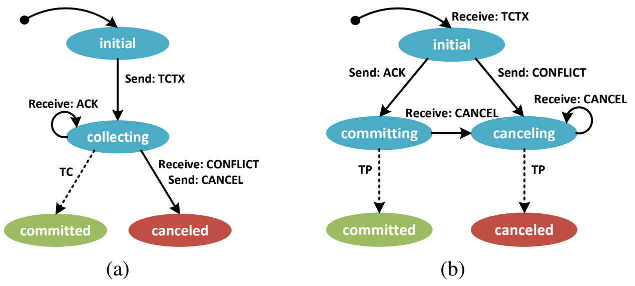


Fig. 1. The state machines of the coordinator and the participant, where a solid line arrow indicates a state transition triggered by a message, a dashed line arrow indicates a state transition triggered by the timer. (a) The state machine of the coordinator, which comprises: four kinds of states: the initial state, the collecting state, the committed state, the canceled state; four kinds of messages being sent or received: the transaction context `TCTX`, the acknowledgment message `ACK`, the conflict message `CONFLICT`, the cancellation message `CANCEL`; the timer of coordinator `TC`. (b) The state machine of the participant, which comprises: five kinds of states: the initial state, the committing state, the canceling state, the committed state, the canceled state; four kinds of messages being sent or received: the transaction context `TCTX`, the acknowledgment message `ACK`, the conflict message `CONFLICT`, the cancellation message `CANCEL`; the timer of participant `TP`.

messages from the participants, the state will not be moved immediately to committed and be maintained in collecting until `TC` is triggered. During the waiting time defined by `TC`, the coordinator collects the messages from the participants. As soon as `TC` is triggered, whether or not all `ACK` messages from the participants are received, the state of the coordinator will change into the committed state.

As shown in Fig. 1b, once the `TCTX` from the coordinator has been received, a sensor node, being the participant, will start its state machine from the initial state, where the timer of participant `TP` is initialized. If no transactional conflict exists at the participant after receiving `TCTX`, an acknowledgment message `ACK` will be sent to the coordinator, and the transition to the committing state will be triggered. If there exists a conflict, the participant will then send a `CONFLICT` message to the coordinator and change the state to canceling. The participant then waits until `TP` is triggered and makes a state transition correspondingly either from committing to committed, or from canceling to canceled. If any `CANCEL` message broadcast by the coordinator is received during this wait, the transition to the canceling state will be automatically triggered.

We use the following algorithms to implement the state machines of the coordinator and the participant. The transaction commit process on the coordinator side is shown in Algorithm 1, and that on the participant side is shown in Algorithm 2.

Theorem 1. *The transaction commit process described by Algorithm 1 and Algorithm 2 ensures transactional consistency.*

Proof. According to the state machines of the coordinator and the participant, the state set of the coordinator is $\{initial, collecting, committed, canceled\}$, while $\{initial, committing, canceling, committed, canceled\}$ is the

Algorithm 1 Transaction commit process of coordinator

```
1:  begin transaction
2:      Initialize WSNT(TID,TCTX,P)        ▷ the initial state
3:      Start TC
4:      Broadcast TCTX to P
5:      Collect messages from P            ▷ the collecting state
6:      if message CONFLICT received then
7:          Broadcast CANCEL to P
8:          Close TC
9:          Abort transaction              ▷ the canceled state
10:     else
11:         Wait till TC triggered
12:         if TC triggered then
13:             Commit transaction         ▷ the committed state
14:         end if
15:     end if
16: end transaction
```

Algorithm 2 Transaction commit process of participant

```
1:  begin transaction
2:      Receive TCTX from coordinator
3:      Start TP                           ▷ the initial state
4:      if no conflict exists then
5:          Send ACK to coordinator
6:          Wait till TP triggered         ▷ the committing state
7:          if CANCEL received then
8:              if TP triggered then       ▷ the canceling state
9:                  Abort transaction      ▷ the canceled state
10:             end if
11:         else
12:             if TP triggered then
13:                 Commit transaction     ▷ the committed state
14:             end if
15:         end if
16:     else
17:         Send CONFLICT to coordinator
18:         Wait till TP triggered         ▷ the canceling state
19:         if TP triggered then
20:             Abort transaction          ▷ the canceled state
21:         end if
22:     end if
23: end transaction
```

state set of the participant. Similar to the idea of use case sequencing proposed by Briand and Labiche [18], all the possible state transition sequences can be written as follows.

The state transition sequences of the coordinator are:

$$\begin{cases} S_c^1 = initial \cdot collecting^* \cdot committed, \\ S_c^2 = initial \cdot collecting^* \cdot canceled. \end{cases} \tag{1}$$

The state transition sequences of the participant are:

$$\begin{cases} S_p^1 = initial \cdot committing \cdot committed, \\ S_p^2 = initial \cdot committing \cdot canceling^* \cdot canceled, \\ S_p^3 = initial \cdot canceling^* \cdot canceled. \end{cases} \tag{2}$$

In (1) and (2) above, the dot operator "$\cdot$" indicates a state transition, e.g., $\mathit{initial} \cdot \mathit{collecting}$ means the transition from the initial state to the collecting state. The star operator "$*$" indicates the self-transitions of a state, meaning a state being repeated for finite times.

With the help of (1) and (2), we are able to list all the possible pairs of the state transition sequences that will happen on the coordinator side and the participant side in the form of two-tuples: (S_c^1, S_p^1), (S_c^1, S_p^2), (S_c^1, S_p^3), (S_c^2, S_p^1), (S_c^2, S_p^2), and (S_c^2, S_p^3).

Among all those six pairs of state transition sequences, some can be proven invalid based on the message triggering mechanism implied in the state machines. Here we use an elimination method to exclude all invalid sequence pairs.

The pair (S_c^1, S_p^2) is invalid. The transition $\mathit{committing} \cdot \mathit{canceling}$ in S_p^2 is triggered by the cancellation message `CANCEL` which is released if and only if $\mathit{collecting} \cdot \mathit{canceled}$ is made at the coordinator. This implicational relation is written as:

$$\mathit{committing} \cdot \mathit{canceling} \rightarrow \mathit{collecting} \cdot \mathit{canceled}. \tag{3}$$

The state transition $\mathit{collecting} \cdot \mathit{canceled}$ does not appear in S_c^1, indicating the implication (3) to be false. Thus, (S_c^1, S_p^2) is invalid.

Similarly, the pair (S_c^1, S_p^3) is invalid. The transition $\mathit{initial} \cdot \mathit{canceling}$ in S_p^3 is triggered by the cancellation message `CANCEL` which is released if and only if $\mathit{collecting} \cdot \mathit{canceled}$ is made at the coordinator. This implicational relation is written as:

$$\mathit{initial} \cdot \mathit{canceling} \rightarrow \mathit{collecting} \cdot \mathit{canceled}. \tag{4}$$

The state transition $\mathit{collecting} \cdot \mathit{canceled}$ does not appear in S_c^1, indicating the implication (4) to be false. Thus, (S_c^1, S_p^3) is invalid.

Lastly, the pair (S_c^2, S_p^1) is invalid. The state transition $\mathit{collecting} \cdot \mathit{canceled}$ in S_p^1 releases the cancellation message `CANCEL` that necessarily triggers either $\mathit{committing} \cdot \mathit{canceling}$ or $\mathit{canceling}^*$ at the participant. Otherwise, the transition $\mathit{committing} \cdot \mathit{canceling}$ or $\mathit{canceling}^*$ also implies the necessity of $\mathit{collecting} \cdot \mathit{canceled}$ to happen at the coordinator. This biconditional relation is written as:

$$\begin{aligned} \mathit{collecting} \cdot \mathit{canceled} \leftrightarrow (&\mathit{committing} \cdot \mathit{canceling}) \\ &\|(\mathit{canceling}^*). \end{aligned} \tag{5}$$

Neither $\mathit{committing} \cdot \mathit{canceling}$ nor $\mathit{canceling}^*$ appears in S_c^2, indicating the biconditional statement (5) to be false. Thus, (S_c^2, S_p^1) is invalid.

The remaining sequence pairs, namely, (S_c^1, S_p^1), (S_c^2, S_p^2), and (S_c^2, S_p^3), are all valid, as no such contradictions are found within them. In all these remaining sequence pairs, both the coordinator and the participant come to the same state, either being committed or being canceled, when the final state is reached. Thus, the transaction commit process provided by our algorithms is proven to be transactionally consistent. $\square$

The interval durations of `TC` and `TP` are set equally so as to ensure that the participants are synchronized with the coordinator. If a node failure occurs to a participant, which makes the coordinator fail to receive any `ACK` or `CONFLICT` message from this certain participant, such a malfunctioning participant node will be immediately eliminated from the current network because of the self-organizing property of WSNs, and thus, data consistency of the transaction process in the working network will not be affected.

C. Concurrency Control Strategy

There are two types of concurrency control strategies in terms of transaction processing: the pessimistic concurrency controller (PCC) and the optimistic concurrency controller (OCC). PCC assumes the worst case to happen and prevents it by locking the record as soon as the record is selected. PCC releases the locked resources only after the whole transaction has finished. OCC, on the contrary, locks the record only when an update takes place. OCC divides an update transaction into three phases: the reading phase where the transaction is allowed to read the data to be updated and tentatively calculate the updating result, the verification phase where any conflict with other active transactions is being detected, and the writing phase where the transaction is committed and the update is carried out based on the pre-calculated result at the reading phase. In WSN systems, locking every data record and sending the locking request to each sensor node is way too expensive to conduct and may also cause deadlocks. Therefore, in our model, we design the concurrency controller based on OCC, which not only has the characteristics of non-blocking and deadlock-free but also computationally cheap for low concurrency systems.

Because of the distributed architecture of WSNs, the concurrency controller of the base station and that of the sensor nodes should be designed separately. In this section, we discuss the concurrency control schemes on the base station side as well as the sensor node side.

1) Concurrency Control on the Base Station Side: The concurrency control scheme on the base station side is explained as follows from the perspectives of three different types of conflicts, namely, the conflict between queries (read-read conflict), the conflict between query and update (read-write conflict), and the conflict between updates (write-write conflict).

a) Read-read controller: Queries are compatible with each other so they do not need to be isolated. However, due to the conflict-prone environment of WSN systems, we suggest coordinating each message being sent to the base station by setting a timer at each sensor node, which makes the message be sent at a regular time interval, so that the message conflict and the packet loss rate can be reduced.

b) Read-write controller: Unlike a traditional query of DBMS, a WSN query is continuous, which poses a higher risk of read-write conflict. It is doable to use some priority-based scheduling strategies [10] to deal with conflicts during a continuous query. However, on account of the unstable communication in WSNs, terminating a running transaction according to priority is dangerous and may result in data inconsistency. Therefore, in our model, it is prohibited for a continuous query to terminate an executing update transaction. We manage the read-write conflict through two queues, an active queue for transactions being executed and a waiting queue for transactions with conflict. If an update transaction starts during a continuous query, a conflict detection will be made to check whether or not the metadata that the update modifies is related to the query. The update transaction will be suspended and be put into the waiting queue if such conflict is detected. And if not, the update transaction will be put into the active queue and get executed. Otherwise, if a query takes place during an update with no conflict, the query will get executed right away. And if there is a conflict, the query will be put into the waiting queue. All the transactions in the waiting queue will be verified and pushed into the active queue sequentially once the current running transaction is finished.

c) Write-write controller: Since most of the WSN applications are developed for querying sensor data, a WSN does not usually have as many update transactions to deal with as compared to query transactions. Besides, considering the network failure and message loss that occasionally occur in WSNs, our model employs a strict control on update transactions at the verification phase on the coordinator side by doing conflict detection only for one update each time. The newly arrived update will be suspended in the waiting queue described in the read-write controller until the current one is committed.

2) Concurrency Control on the Sensor Node Side: Sensor nodes, as low-end devices, are only responsible for receiving queries and sending data, without complicated tasks to do in WSNs. Hence, our model only deals with the write-write conflict that could take place on the sensor node side.

a) Write-write controller: In WSNs, a sensor node is allowed to change its metadata information by itself, e.g., the node reduces its sampling rate by half in low power mode. Such self-adjustment may conflict with a global WSN update started by the coordinator. The write-write conflict at sensor nodes is getting controlled in the following manner. If a sensor node receives the transaction context `TCTX` of a global update during self-adjustment, verification will be made to detect the conflict between the self-adjustment and the global update by checking whether or not they are modifying the same metadata. The global update will be carried out as long as no conflict is detected. If the conflict exists, this certain sensor node will send a `CONFLICT` message to the coordinator, triggering the coordinator to broadcast the `CANCEL` message to all participants. The global update will then be canceled.

III. Implementation Framework

Based on the atomic transaction process described in Section II-B and the concurrency controllers described in Section II-C, we design a whole transaction processing framework for the implementation of STCP, which consists of five modules, the transaction manager, the concurrency controller, the active queue, the waiting queue, and the transaction processing module. Fig. 2 gives an overview of the interactions between the five modules.

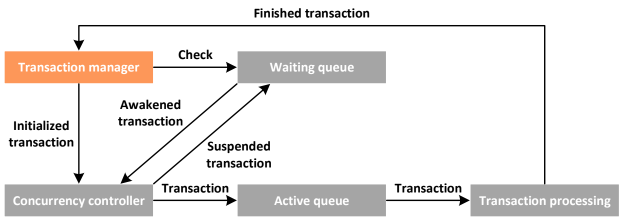


Fig. 2. The transaction processing framework.

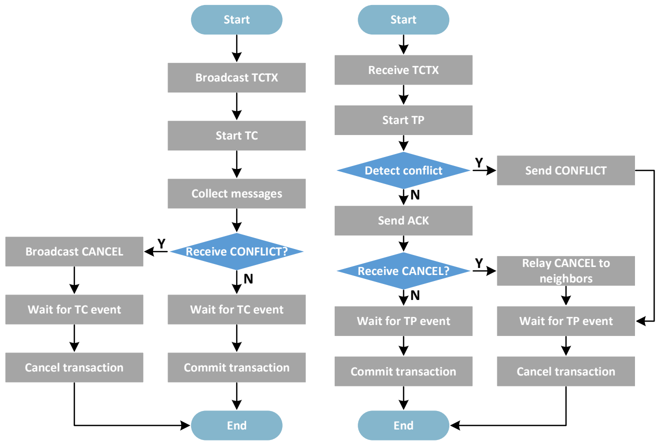


Fig. 3. The transaction processing (a) on the coordinator side and (b) on the participant side.

The transaction manager, the core component of our framework, is responsible for managing the life cycle of a transaction which includes the initialization, the scheduling, and the destruction of a transaction.

The concurrency controller is the implementation of a set of concurrency processing interfaces based on the design in Section II-C. Once a new WSN transaction is initialized, the transaction manager will call the concurrency controller to do conflict detection for it. The transaction with no detected conflict will be put into the active queue and get executed right away. The transaction with conflict will be suspended in the waiting queue, waiting to be woken up by the transaction manager.

The active queue and the waiting queue, as mentioned in Section II-C1, are designed for concurrency management. The active queue stores the active transactions, while the waiting queue is responsible for storing the suspended transactions.

The transaction processing module comprises all transactional processes that take place at the coordinator and the participants after a transaction is getting verified by the concurrency controller, as shown in Fig. 3a and Fig. 3b. The implementation of this module is based on the design of the atomic commit algorithm in Section II-B. Moreover, in order to enhance the robustness of our implementation, upon receiving the `CANCEL` message from the coordinator,

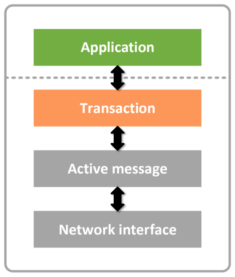


Fig. 4. The software stack of the implementation of STCP.

each participant is forced to relay the `CANCEL` message to its neighbors. As soon as a transaction is committed or canceled, the transaction will be recollected by the transaction manager and then be destroyed.

The finish of each transaction triggers the transaction manager to check the waiting queue and wake up the first transaction in it. The awakened transaction will be called by the concurrency controller and get restarted from the conflict verification phase.

IV. Simulation and Results

In this section, we briefly describe the setup of the simulation experiment and the experimental results of the implementation of STCP.

A. Experimental Environment

We implement STCP according to the framework as explained in Section III using the nesC language and test our implementation using the TOSSIM simulator in TinyOS. The implementation environment can be visualized in the four-layer stack structure illustrated in Fig. 4.

The network interface layer and the active message layer are infrastructure layers provided by TOSSIM. The former simulates the radio communication that occurs at the physical layer of WSNs, and the latter, built upon the network interface layer, provides a set of network communication interfaces implementing the WSN message mechanism. The transaction layer is where STCP is implemented, based on the inter-node communication provided by the active message layer. The application layer is developed for testing the application scenarios of STCP, where we use Python scripts to simulate the transaction communication between the base station and the sensor nodes.

We simulate a WSN system installed in a $20\,\mathrm{m} \times 20\,\mathrm{m}$ factory workshop with eleven nodes randomly spread in it, of which one is the base station, and the other ten the sensor nodes. To improve the quality of radio frequency (RF) simulation, we simulate the hardware noise floor for each node as $-98.0\,\mathrm{dBm}$, with a standard deviation of $4.0\,\mathrm{dBm}$ white Gaussian noise, to produce a topological model that describes the signal gains between each two nodes, which is shown in Fig. 5 in the form of heat map matrix, and we use this model to feed the closest pattern matching (CPM) algorithm

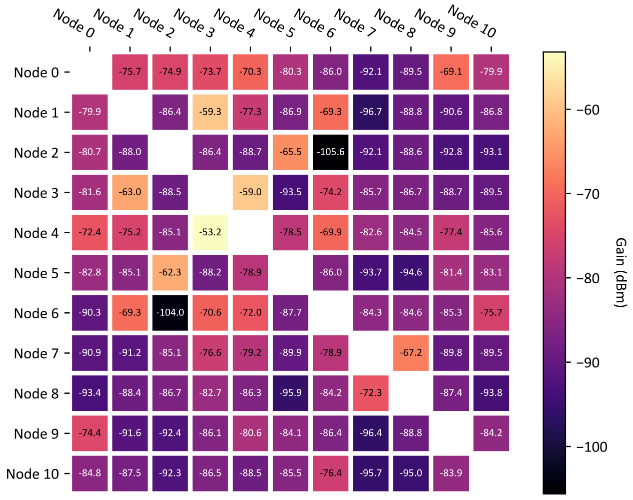


Fig. 5. The topology of the experimental WSN system.

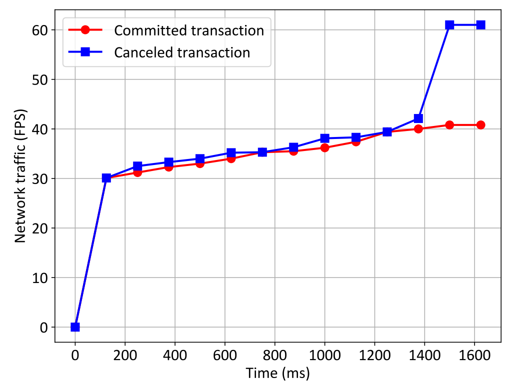
that generates a statistical noise model for each sensor node in the experimental WSN system.

B. Simulation Results

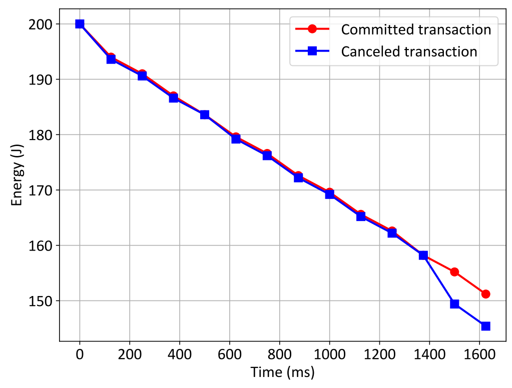
To analyze the performance of our protocol, we compare the average network traffic and the average energy consumption in the experimental network between a committed transaction and a canceled transaction processed by STCP over the execution time of one transaction, as shown in Fig. 6. Fig. 6a shows the average network traffic of each sensor node within one transaction period and Fig. 6b shows the average remaining energy of each sensor node within one transaction period. The time interval of the timer is preset as $1650\,\mathrm{ms}$ and each sensor node is assigned with an initial energy of $200\,\mathrm{J}$. It can be interpreted from Fig. 6a and Fig. 6b that, during most of the transaction period, the network traffic and energy consumption increase gradually and remain on the same level for both the committed and the canceled transactions. A $42.5\,\%$ increase of network traffic and an $11.9\,\%$ increase of energy consumption occur at the final stage of the canceled transaction where the `CANCEL` message is broadcast by the base station, which is reasonable and acceptable.

We also compare the energy consumption caused by STCP with that caused by the traditional 2PC protocol. We process ten transactions (five being committed and five being canceled) using STCP and 2PC respectively and calculate the average network energy consumption during the transactions. Fig. 7 shows the average remaining energy of each sensor node during one transaction period processed by STCP and 2PC. An instance of network remaining energy, in which no transaction is processed, is given in Fig. 7 as a comparative baseline, reflecting the normal energy loss in the experimental network.

It can be seen from Fig. 7 that 2PC causes much more (around $62.8\,\%$) energy loss than STCP during the process of one transaction, due to the complicated interactions required by 2PC between the coordinator and the participants, especially at the final stage of a transaction, where a higher



(a)



(b)

Fig. 6. (a) The network traffic and (b) the network energy consumption of committed and canceled transactions.

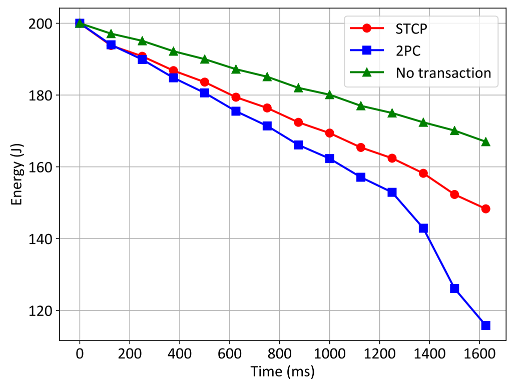


Fig. 7. The comparison of energy consumption between STCP and 2PC.

message retransmission rate usually takes place. However, STCP simplifies these interactions through a timer that triggers the commitment or cancellation of a transaction, thereby better satisfying the energy demand of WSNs.

V. Conclusion

In this paper, we present STCP, a completely defined transaction processing protocol for WSN systems, which adapts better to the demand and characteristics of WSNs than the existing atomic commit protocols of distributed systems. We introduce the theoretical design as well as an implementation framework of STCP. STCP is capable of effectively and efficiently dealing with data loss and data conflict issues to ensure data consistency in WSN systems, showing potential in future WSN applications.

Further research can be made on STCP with networks having more complicated structures, such as cluster-based networks, so as to find more possible applications of STCP in large-scale WSN systems.

References

[1] B. Yin, S. Zhou, S. Zhang, K. Gu, and F. Yu, "On efficient processing of continuous reverse skyline queries in wireless sensor networks," *KSII Trans. Internet Inf. Syst.*, vol. 11, no. 4, pp. 1931–1953, 2017.

[2] S. Yang, "IoT stream processing and analytics in the fog," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 21–27, 2017.

[3] K. Wang, Y. Shao, L. Shu, C. Zhu, and Y. Zhang, "Mobile big data fault-tolerant processing for eHealth networks," *IEEE Netw.*, vol. 30, no. 1, pp. 36–42, 2016.

[4] S. Randhawa and S. Jain, "Data aggregation in wireless sensor networks: Previous research, current status and future directions," *Wireless Pers. Commun.*, vol. 97, no. 3, pp. 3355–3425, 2017.

[5] D. Izadi, J. H. Abawajy, S. Ghanavati, and T. Herawan, "A data fusion method in wireless sensor networks," *Sensors*, vol. 15, no. 2, pp. 2964–2979, 2015.

[6] Y. Yao and J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, 2002.

[7] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[8] R. Khoury, T. Dawborn, B. Gafurov, G. Pink, E. Tse, Q. Tse, K. Almi'Ani, M. Gaber, U. Röhm, and B. Scholz, "Corona: Energy-efficient multi-query processing in wireless sensor networks," in *Proc. 15th Int. Conf. on Database Systems for Advanced Applications*, Tsukuba, Japan, Apr. 2010, pp. 416–419.

[9] A. Belfkih, C. Duvallet, B. Sadeg, and L. Amanton, "A real-time query processing system for WSN," in *Proc. 16th Int. Conf. on Ad Hoc Networks and Wireless*, Messina, Italy, Sep. 2017, pp. 307–313.

[10] L. Gürgen, C. Roncancio, C. Labbé, and V. Olive, "Transactional issues in sensor data management," in *Proc. 3rd Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2006*, Seoul, Korea, Sep. 2006, pp. 27–32.

[11] C. Reinke, N. Hoeller, and V. Linnemann, "Adaptive atomic transaction support for service migration in wireless sensor networks," in *Proc. 7th Int. Conf. on Wireless and Optical Communications Networks*, Colombo, Sri Lanka, Sep. 2010, pp. 1–8.

[12] C. Reinke, N. Hoeller, J. Neumann, S. Groppe, V. Linnemann, and M. Lipphardt, "Integrating standardized transaction protocols in service-oriented wireless sensor networks," in *Proc. 2009 ACM Symp. on Applied Computing*, Honolulu, HI, USA, Mar. 2009, pp. 2202–2203.

[13] S. Obermeier, S. Böttcher, and D. Kleine, "CLCP – A distributed cross-layer commit protocol for mobile ad hoc networks," in *Proc. 2008 IEEE Int. Symp. on Parallel and Distributed Processing with Applications*, Sydney, NSW, Australia, Dec. 2008, pp. 361–370.

[14] C. Reinke, N. Hoeller, J. Neumann, S. Groppe, S. Werner, and V. Linnemann, "Analysis and comparison of atomic commit protocols for adaptive usage in wireless sensor networks," in *Proc. 2010 IEEE Int. Conf. on Sensor Networks, Ubiquitous, and Trustworthy Computing*, Newport Beach, CA, USA, Jun. 2010, pp. 138–145.

[15] C. Reinke, "Adaptive service migration and transaction processing in wireless sensor networks," in *Proc. 7th Middleware Doctoral Symp.*, Bangalore, India, Nov. 2010, pp. 8–13.

[16] Y. Liu, K. Wang, Y. Lin, and W. Xu, "LightChain: A lightweight blockchain system for industrial internet of things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3571–3581, 2019.

[17] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 870–882, 2019.

[18] L. Briand and Y. Labiche, "A UML-based approach to system testing," *Softw. Syst. Model.*, vol. 1, no. 1, pp. 10–42, 2002.