RTSP: An Accurate and Energy-Efficient Protocol for Clock Synchronization in WSNs

Muhammad Akhlaq, *Member, IEEE*, and Tarek R. Sheltami, *Member, IEEE*

***Abstract*—Wireless sensor networks need accurate time synchronization for data consistency and coordination. Although the existing algorithms for time synchronization offer very good accuracy, their energy consumption is high, and distant nodes are poorly synchronized. We propose a Recursive Time Synchronization Protocol (RTSP) which accurately synchronizes all the nodes in a network to a global clock using multi-hop architecture in an energy-efficient way. It achieves better performance due to the MAC-layer time-stamping based on Start of Frame Delimiter byte, infrequent broadcasts by a dynamically elected reference node, compensation of the propagation delay and adjustment of the timestamps at each hop, estimation of the relative skew and offset using least square linear regression on two data points (2LR), adaptive re-synchronization interval, aggregation of the synchronization requests, and energy awareness. A detailed analysis of the sources of errors is also provided. Simulation results show that the RTSP can achieve an average accuracy of 0.3 microseconds in a large multi-hop flat network while using five-times lesser energy than that of FTSP in the long run and performs even better in a clustered network where it can achieve an average accuracy of 0.23 microseconds while using seven-times lesser energy.**

***Index Terms*—Accuracy, algorithm, clock, energy efficiency, estimation, protocol, synchronization.**

I. Introduction

WIRELESS Sensor Networks (WSNs) require accurate time synchronization, usually less than one microsecond, for many reasons, such as precise time-stamping of messages, in-network signal processing, time-based localization, TDMA-based medium access, cooperative communication, coordinated actuation, and energy-efficient duty cycling of sensor nodes. There are several ways to achieve synchronization, which include ordering of events, local synchronization, global synchronization, etc. However, many applications need global time synchronization which requires all nodes to be synchronized with one global clock.

As WSNs are commonly deployed for monitoring the remote and hazardous environments, it is not feasible to recharge or replace the battery of sensor nodes. Therefore, sensor nodes should use their energy very efficiently even when a power harvesting technique is used with the battery of limited capacity. It is important to note that most of the sensors' energy is consumed by message transmission and idle listening, and that clock synchronization algorithms work by exchanging messages among the nodes. Therefore, the lifetime of WSNs can be significantly prolonged by using an energy-efficient protocol for clock synchronization. However, the total energy consumed by time-synchronization protocol depends on the re-synchronization interval T and the ratio of time-synchronization messages to the total number of messages in the network that is determined by the application. A detailed study of other techniques for energy conservation in WSNs is provided by [1].

The issue of clock synchronization has been investigated extensively, and several methods or protocols have been proposed for global time synchronization, such as GPS-based clock synchronization, Network Time Protocol (NTP) [2], Precision Time Protocol (PTP) [3], Reference Broadcast Synchronization (RBS) [4], Timing-sync Protocol for Sensor Networks (TPSN) [5], and Flooding Time Synchronization Protocol (FTSP) [6].

The GPS-based clock-synchronization [7] can provide an accuracy of 1 μs or better, but it is costly, energy-inefficient, and infeasible in obstructed environments. The NTP [2] is commonly used in traditional computer networks including the Internet, but it is not suitable for WSNs because of its very low accuracy (i.e., in the order of milliseconds only), high complexity, and energy inefficiency [4]–[6], [8]. The PTP, defined by IEEE 1588 standard, can achieve clock accuracy in the sub-microsecond range for networked measurement and control systems [9]–[11], but it is suitable only for hierarchical master-slave architecture. The RBS uses receiver-receiver synchronization to produce an average accuracy of 29.1 μs for a single hop network, but this accuracy is not sufficient for WSNs which require an accuracy of 1 μs or better. The TPSN uses sender-receiver synchronization and MAC-layer time-stamping of messages at the sender side to provide an average accuracy of 16.9 μs for a single hop network and less than 20 μs for multi-hop network, but it still not sufficient for WSNs.

The FTSP [6] is the most commonly used protocol for clock synchronization in WSNs. It broadcasts messages with timing information from a single sender to several receivers without any exchange among themselves, strives to tackle the flaws of TPSN. It dynamically elects a reference node which regularly floods its current timestamp into the network creating an ad-hoc tree structure of the network instead of a fixed spanning tree. The MAC-layer time-stamping at both sender and receiver sides eliminates all kind of random delays except propagation delay. The timestamps are made at the end of each byte after Start of Frame Delimiter (SFD) or SYNC byte, normalized, averaged,

Manuscript received April 6, 2012; revised July 24, 2012; accepted September 10, 2012. Date of publication January 4, 2013; date of current version February 5, 2013. This work was funded by King Fahd University of Petroleum and Minerals, and King Abdulaziz City for Science and Technology, Saudi Arabia, under project # AR-29-71. The Associate Editor coordinating the review process for this paper was Dr. Deniz Gurkan.

The authors are with the College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia (e-mail: akhlaq@kfupm.edu.sa; tarek@kfupm.edu.sa).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIM.2012.2232472

error corrected, and then final timestamp is embedded into the message. A node waits for sufficient data points that are pairs of global and local timestamps and then estimates the offset and skew using least square linear regression. Any node that is fully synchronized with the reference node begins flooding its own estimation of the global clock. In this way, the FTSP provides an accuracy of 1.48 μs for single hop case and about 0.5 μs per hop in a multi-hop network [6].

There have been many efforts to improve the FTSP in terms of accuracy, efficiency, energy consumption, etc. For example, D. Cox *et al.* [12] have improved the accuracy and power consumption in a single hop network using a different method of time-stamping that is based on the SFD byte. In the SFD-based time-stamping, messages are time-stamped using the time at which radio chip generates an interrupt for the microcontroller after the SFD byte has been sent or received. Similarly, M. Aoun *et al.* [13] have obtained an accuracy of 0.4 μs in a single hop network by using SFD-based time-stamping along with Kalman-filter for skew estimation. Although the accuracy of FTSP and its improved versions is sufficiently good, the energy consumption is very high and the distant nodes are poorly synchronized.

We have proposed a Recursive Time Synchronization Protocol (RTSP) [14] that was presented at the IEEE SAS2012 in Brescia, Italy. The RTSP provides an average accuracy of 0.3 μs per-hop in a large multi-hop flat network (i.e., without clustering) while using only 1/5th of the energy consumed by FTSP in the long run. However, as WSNs are usually clustered and hierarchical, we have modified and extended the RTSP algorithm to make it work with clustered networks as well. This paper extends the RTSP algorithm for clustered networks. In case of non-clustered or flat network, each node is assumed to be a clusterhead in order to run RTSP algorithm correctly. Simulation results show that the extended RTSP algorithm further improves the accuracy and energy consumption, i.e., it can provide an average accuracy of 0.23 μs in a large multi-hop clustered network while using only 1/7th of the energy consumed by FTSP in the long run.

The rest of this paper is organized as follows. Section II describes the system model while Section III explains the RTSP algorithm in detail. Section IV analyzes the sources of errors, while Section V analyzes the sources of efficiency. The performance evaluation of RTSP compared to existing algorithms is discussed in Section VI. Finally, Section VII concludes the paper and describes the possible future work.

II. System Model for RTSP

First, we briefly describe the system model, assumptions, time-stamping, message structure, and recursion in RTSP.

A. Clock Model

Each sensor node has a hardware clock consisting of timer circuitry, which is usually based on quartz crystal oscillator and hence unstable and error prone. The hardware clock can be translated into logical clock which is used for time keeping.

The hardware clock with instantaneous oscillator frequency $f_i(t)$ at an ideal time t, is defined as (1), where f_0 is the ideal frequency, Δf is the frequency offset, d_f is the drift in frequency, and $r_f(t)$ is the random error process

$$f_i(t) = f_0 + \Delta f + d_f t + r_f(t). \tag{1}$$

Assuming $t = 0$ as the initial reference time, the related logical clock reads time $C_i(t)$ at ideal time t, defined as

$$C_i(t) = C_i(0) + \frac{1}{f_0} \int_0^t f_i(t) dt. \tag{2}$$

We can obtain expression for the time $C_i(t)$ of clock i at a given ideal time t by combining (1) and (2), as follows:

$$C_i(t) = C_i(0) + \frac{1}{f_0} \int_0^t \left(f_0 + \Delta f + d_f t + r_f(t)\right) dt. \tag{3}$$

To derive a simple linear model for non-ideal clock, we can assume that there is no frequency drift (d_f) and that random clock error $(r_f(t))$ is a zero mean [15]. Therefore, after elimination of the last two terms in (3) and further simplification, we get $C_i(t) = C_i(0) + (1 + (\Delta f / f_0))t$, which can be written in very simple form as (4), where α_i is the clock offset (i.e, the deviation from ideal time) at the reference time $t = 0$ and β_i is the clock skew (i.e., the frequency of a clock)

$$C_i(t) = \alpha_i + \beta_i t. \tag{4}$$

Clocks are usually specified with a maximum drift rate ρ (i.e., the rate of change of frequency with respect to the ideal clock), such that $1 - \rho \leq \beta_{\mathrm{i}} \leq 1 + \rho$. That is, the clocks of any two perfectly synchronized nodes may drift away from each other at a rate at most 2ρ. Therefore, in order to keep their relative offset within δ seconds all the time, they need to be resynchronized within $\delta/2\rho$ seconds, i.e., re-synchronization interval $\mathrm{T} \leq \delta/2\rho$. However, frequent re-synchronizations are not feasible due to inefficiency, cost, etc. Therefore, it is essential to estimate the drift or skew in order to keep the sensor clocks synchronized without any need for frequent re-synchronizations.

B. Assumptions

Following assumptions, which are realistic and commonly found in literature, are made in the proposed algorithm:

1) Sensor nodes are uniquely identified by their numeric IDs from 0 to n–1.
2) Time-stamping of messages at MAC-layer is possible for each node.
3) Neighboring nodes can communicate over an unreliable and error-corrected wireless channel.
4) Broadcasting of messages is possible in the network.
5) Skew and connectivity do not change during the short interval between synchronization request and reply.
6) Propagation delay in one direction is exactly equal to the other.
7) A simple linear relationship exists between the clocks of two sensor nodes in a short duration.

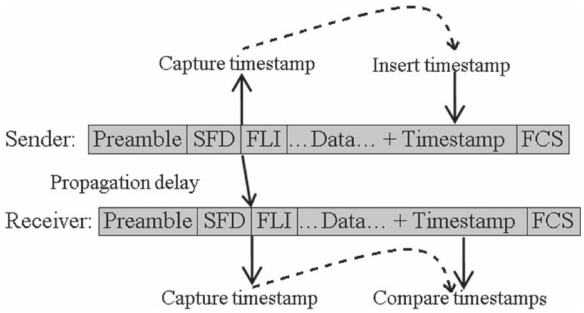

Fig. 1. Structure of IEEE 802.15.4 MAC frame and the time-stamping.

C. Time-Stamping

MAC-layer time-stamping of messages [5], [12], [13], [16], both at sender and receiver sides, is very useful for time synchronization in WSNs. The standard IEEE 802.15.4 compliant frame consists of 4 bytes of preamble, 1 byte of SFD, 1 byte of frame length indicator, data payload of 125 bytes maximum, and 2 bytes of frame check sequence [17]. Fig. 1 shows that the RTSP algorithm, like D. Cox *et al.* [12] and M. Aoun *et al.* [13], timestamps messages when radio chip generates an interrupt for the microcontroller after the SYNC/SFD byte has been sent or received. Ignoring the propagation delay, this interrupt occurs almost simultaneously both at the sender and receiver chips with a difference of 3 μs. Therefore, the timestamp at sender side is incremented by 3 μs.

This time-stamping approach is simpler, faster, and more accurate than that of FTSP in which timestamps are made at the end of each byte after SFD byte, normalized, averaged, error corrected, and then final timestamp is embedded into the message. It is important to note that this approach is also unable to compensate the propagation delay. Therefore, we introduce a mechanism for finding and compensating the propagation delays in order to achieve higher accuracy of time synchronization in the RTSP algorithm.

D. Structure of the RTSP Messages

The RTSP algorithm achieves time synchronization by exchanging messages among nodes. There are three types of messages: ERN (enquiry/election of the reference node), REQ (request for time synchronization), and REP (reply for time synchronization). The structure of an RTSP message is shown in Table I. Note that an RTSP-ERN message with a negative value for the refNodeID field is treated as an enquiry, while a non-negative value is treated as an announcement or contest. A node requests timing information by sending an RTSP-REQ message to the reference node, while the reference node or any synchronized node on the request-path replies with an RTSP-REP message. The fields T1, T2, T3, and Tr are used for timestamps after SFD byte has been sent, received, or forwarded.

E. Recursion and Multi-hop Synchronization

The RTSP algorithm does not require any tree-like structure; it works with any network topology, flat and clustered. A time

TABLE I
STRUCTURE OF THE RTSP MESSAGES

Field	Description
msgType	Type of the message: ERN (enquiry/election of the reference node), REQ (request for time synchronization), and REP (reply for time synchronization).
msgID	ID of the message.
originID	ID of the node that originated the message.
imSrcID	ID of immediate source node that forwarded the message.
imDestID	ID of immediate destination for message forwarding.
refNodeID	ID of the reference node known (-1 for ERN enquiry).
T1	Local time when the message was sent or forwarded.
T2	Local time when the message was received.
T3	Local time when reply was sent or forwarded.
Tr	Reference time when the reply was sent or forwarded.

Note: 1) We use 0 for Null value, and * for broadcast address.
2) RTSP-REQ msg takes Null values for T2, T3 and Tr.

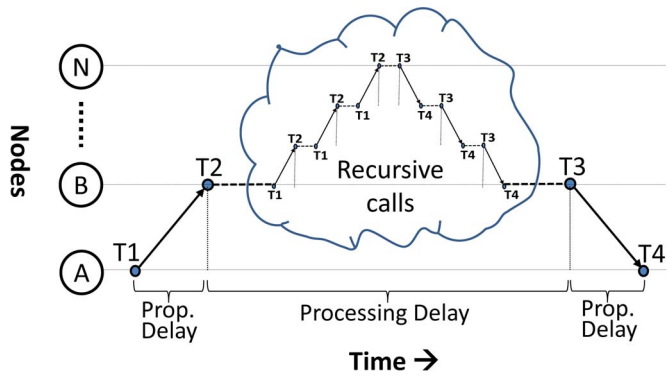

Fig. 2. Request-and-reply mechanism in RTSP.

synchronization request can be initiated by any node in the network, which is recursively forwarded to the reference node. The reference node, or any synchronized node on the request-path, replies with timing information that is sent back to the requesting node through the same path. It is important to note that the propagation delay is also compensated at each hop before forwarding the reply.

Fig. 2 explains the request-and-reply mechanism in RTSP. The node A in a multi-hop network sends request at T1 to the reference node via another node B that receives it at time T2. The node B, if not synchronized, forwards this request to another node in a recursive manner until it reaches the reference node or a fully synchronized node N which replies with the timing information that is forwarded to node B through the same path. The node B at time T3 forwards this reply to node A which receives it at time T4. S. Ganeriwal *et al.* [5] have already proved that the performance of TPSN is insensitive to any variations in processing delay, i.e., the interval between T2 and T3 does not affect the accuracy of time synchronization. This gives us an opportunity to make recursive calls during the interval between T2 and T3 in the RTSP algorithm without losing performance.

Moreover, the nodes closer to the reference node are expected to do the following actions before the distant nodes: 1) receive announcement about the newly elected reference

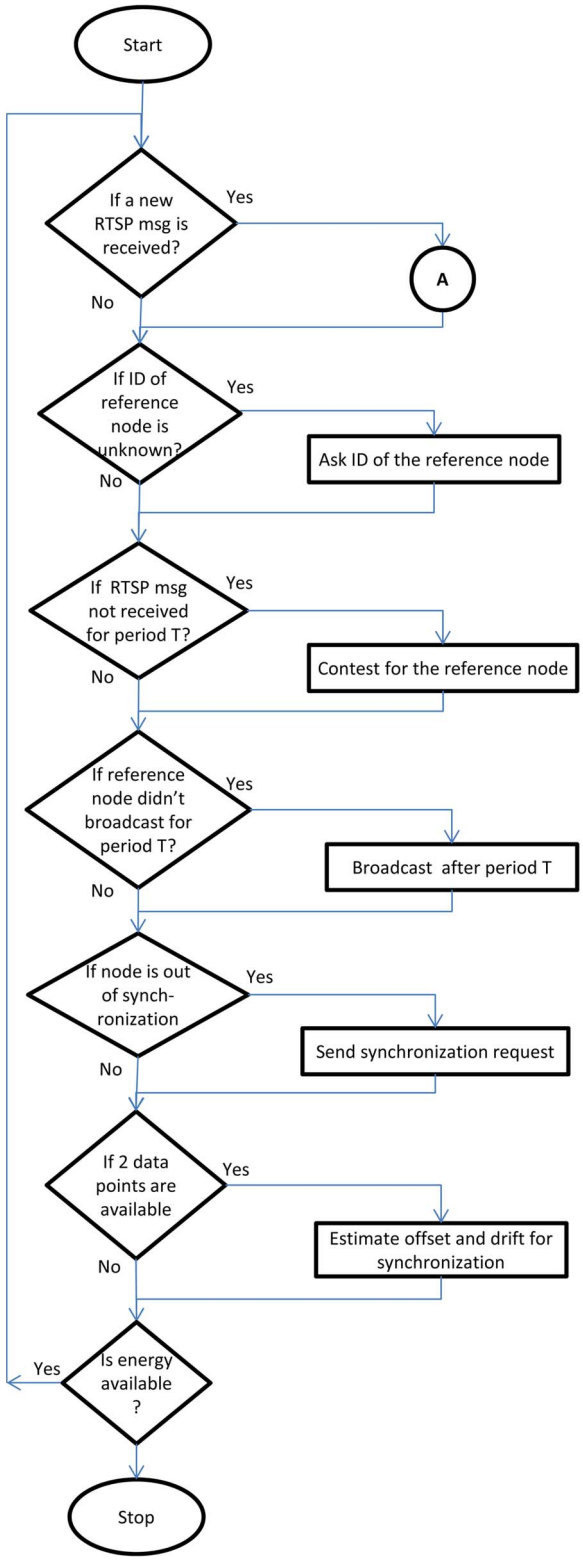

Fig. 3. Recursive Time Synchronization Protocol (RTSP).

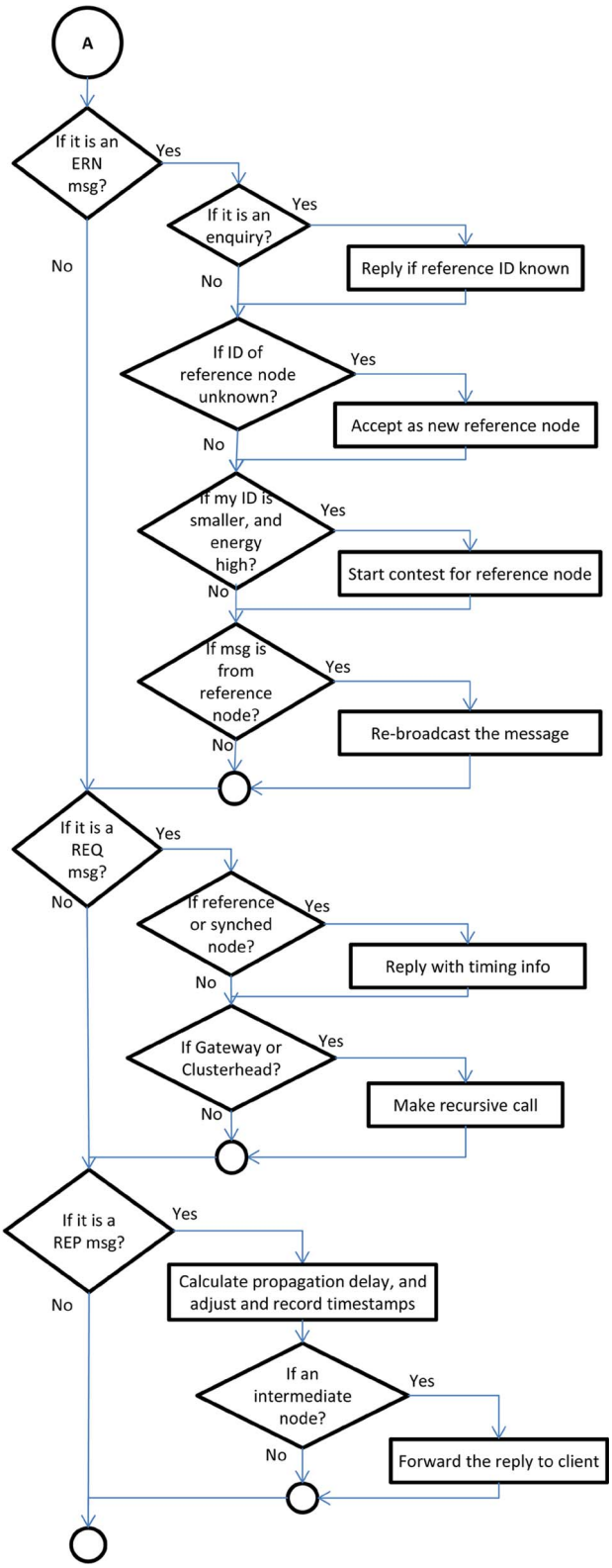

Fig. 4. Segment-A of the RTSP algorithm.

node, 2) initiate the time synchronization request on knowing ID of the reference node, and 3) become synchronized with the reference node. As the closer nodes are expected to become synchronized before distant nodes and any synchronized node on the request-path can reply, very few time synchronization requests are forwarded all the way to the reference node.

III. RTSP

After a WSN boots up, the sensor nodes broadcast an RTSP-ERN enquiry message to ask their neighbors about the identification of reference node, wait for the period T or until a reply is received, run repeatedly the RTSP algorithm that takes care of the dynamic election of a single reference node with

the smallest ID and compensation of the offset and drift. This process is outlined in the Algorithm 1: RTSP and flowcharted in Figs. 3 and 4.

Each node maintains a few variables including myRef for ID of the reference node, and myClient for IDs of the nodes sending time synchronization request through it.

Note that in case of non-clustered or flat network, each node is assumed to be a clusterhead in order to run RTSP algorithm.

The RTSP algorithm is responsible for two major functions, 1) election of the reference node, 2) compensation of the offset and drift, which are explained in the next two subsections.

A. Election of the Reference Node

The RTSP algorithm dynamically elects a single reference node as follows:

1) After startup, a sensor node waits for short time and then sends an RTSP-ERN message containing "−1" value for the refNodeID field in order to ask the neighboring nodes for ID of the reference node (Algorithm 1: RTSP, line 71–77). It is important to note that a clusterhead broadcasts the enquiry message, but a nonclusterhead sends the enquiry message to its own clusterhead. The enquirer node waits for the duration T or until a reply is received, and then acts as follows.
   a) If it does not receive any reply, it enters into the contest for new reference node by broadcasting an RTSP-ERN message by putting its own identification in the refNodeID field (Algorithm 1: RTSP, line 78–85). Note that a node that is not a clusterhead or is low in energy cannot take part in the contest for reference node.
   b) However, if it receives a reply, it saves the identification of reference node in a local variable called myRef, and then broadcasts the RTSP-ERN message (Algorithm 1: RTSP, line 12–18).
2) If a new RTSP-ERN message is received and authenticated (Algorithm 1: RTSP, line 1–4), it checks the value of refNodeID field to do the following.
   a) If the refNodeID field contains any negative value, the message is treated as an enquiry. If a node knows the identification of reference node, it replies directly to the enquirer (Algorithm 1: RTSP, line 6–11).
   b) However, if the refNodeID field contains any nonnegative value, the message is treated as an announcement or contest, and is flooded as follows.
      i) If the receiving node does not know the identification of reference node or it knows some identification that is greater than refNodeID, it learns the identification of new reference node by updating myRef variable and then rebroadcasts the message (Algorithm 1: RTSP, line 12–18).
      ii) If the ID of receiving node is smaller than the current reference node and the receiving node is a clusterhead with sufficient energy, it contests for reference node by broadcasting an RTSP-ERN message (Algorithm 1: RTSP, line 19–24).
      iii) If a clusterhead or a gateway already knows the same ID of reference node, it just rebroadcasts the message (Algorithm 1: RTSP, line 25–32).

The elected reference node takes responsibility of broadcasting the timing-information periodically (Algorithm 1: RTSP, line 86–92). However, the frequency of broadcasting is much lower, i.e., 1/10th of the FTPS because RTSP does not rely on this broadcast for the compensation of skew and offset. Instead, the RTSP algorithm uses this broadcasting mainly for announcing the identification of reference node. Any node that receives a new RTSP-ERN message from the reference node accepts it and then rebroadcasts to its neighbors. However, a duplicate message is ignored or rejected by the receivers.

It is possible that more than one node declare themselves as reference nodes simultaneously. To avoid any conflict, a reference node retreats to an ordinary node on receiving an RTSP-ERN message from another reference node with a smaller identification.

Although an RTSP-ERN message is mainly used to elect and announce the reference node in a network, it is also used to find if a node is losing its accuracy of time synchronization and should request timing information using the RTSP algorithm (Algorithm 1: RTSP, line 93–101).

B. Offset and Drift Compensation

In order to compensate for the offset and drift, and to synchronize all the nodes on request-path to the reference node, the RTSP algorithm (Algorithm 1) works as follows.

1) A node checks the type of newly received RTSP message.
2) If an RTSP-REQ message is received, it notes the receive time T2. If it is a reference node or a synchronized clusterhead or gateway node, it replies with an RTSP-REP message containing timestamps T1, T2, T3, and Tr along with other information (Algorithm 1: RTSP, line 33–42). However, an unsynchronized intermediate node recursively forwards the request to the reference node after saving ID of its client in myClient, and the values of T1 and T2 in T1old and T2old, respectively (Algorithm 1: RTSP, line 43–51).
3) However, if an RTSP-REP message is received, it calculates the value of propagation delay by (5).

$$d = \frac{(T2 - T1) + (T4 - T3)}{2}. \tag{5}$$

Then, it finds the new value of Tr by adding d to the received value of Tr as given by (6).

$$Tr = Tr + d. \tag{6}$$

Then it records the global and local timestamps where global=Tr and local=T4. Moreover, if the node is an intermediate node, it retrieves the values of T1 and T2, calculates the value of Tr by adding to it the elapsed time since T4 using (7), and then forwards the reply to its client node, if any (Algorithm 1: RTSP, line 52–70).

$$Tr = Tr + (T3 - T4). \tag{7}$$

Algorithm 1: RTSP

```
1.  Check if new RTSP msg is received (or overheard);
2.  // on receiving a new message, do the following
3.  If a new authenticated msg is received then
4.    If msg→msgType == "ERN" then
5.      T2 = timestamp after SFD byte received;
6.      If msg→refNodeID<0 AND myRef!=Null then
7.        // an enquiry; reply if reference ID known
8.        T3 = timestamp after SFD byte sent;
9.        RTSP("ERN", msg→msgID, myID, myID,
10.             msg→imSrcID, myRef, msg→T1,
11.             T2, T3, Tr);
12.     Else if (msg→refNodeID >=0 AND
13.           myRef is empty) OR
14.           myRef > msg→refNodeID then
15.       //reference ID unknown/expired; accept new
16.       myRef = msg→refNodeID;
17.       RTSP("ERN",msg→msgID, msg→originID,
18.            myID,*, msg→refNodeID,0,0,0, Tr);
19.     Else if myID < msg→refNodeID AND I'm a
20.          clusterhead AND myEnergy>low then
21.       //smaller ID & fair energy; enter into contest
22.       myRef = myID;
23.       RTSP("ERN", msgID, myID, myID, *,
24.            myID, 0, 0, 0, Tr);
25.     Else if myRef == msg→refNodeID AND
26.          I'm a gateway or a clusterhead AND
27.          myID != msg→originID AND
28.          msg→msgID is new then
29.       // new msg from reference node; rebroadcast
30.       RTSP("ERN",msg→msgID, msg→originID,
31.            myID,*,msg→refNodeID, 0,0,0,Tr);
32.     End if
33.   Else if msg→msgType == "REQ" then
34.     T2 = timestamp after SFD byte received;
35.     If myID==refNodeID OR I'm a synchronized
36.          gateway or clusterhead then
37.       // reply as reference or a synchronized node
38.       T3 = timestamp after SFD byte sent;
39.       Tr = T3;
40.       RTSP("REP",msg→msgID, msg→originID,
41.            myID, msg→imSrcID, myID,
42.            msg→T1, T2, T3, Tr);
43.     Else if I'm a gateway or clusterhead then
44.       //save T1, T2, source, and do a recursive call
45.       T1old = msg→T1;
46.       T2old = T2;
47.       myClient = msg→imSrcID;
48.       T1 = timestamp after SFD byte sent;
49.       RTSP("REQ",msg→msgID, msg→originID,
50.            myID, myRef, myRef, T1, 0, 0, Tr);
51.     End if
52.   Else if msg→msgType == "REP" then
53.     // calculate d, adjust & record timestamp
54.     T4 = timestamp after SFD byte received;
55.     d = ((msg→T2–msg→T1)+(T4–msg→T3))/2 ;
56.     Tr = msg→Tr + d;
57.     Record timestamps global=Tr & local=T4;
58.     If myID != msg→originID AND
59.          myClient not empty then
60.       // recover T1 & T2; and forward the msg
61.       T1 = T1old; T2 = T2old;
62.       T3 = timestamp after SFD byte sent
63.       Tr = Tr+(T3–T4); //+ elapsed time since T4
64.       RTSP("REP",msg→msgID, msg→originID,
65.            myID, myClient, msg→refNodeID,
66.            msg→T1, T2, T3, Tr);
67.       Clear the variable myClient;
68.     End if
69.   End if
70. End if
71. //ask the id of reference node if not known already
72. If myRef == Null AND I'm a clusterhead then
73.   RTSP("ERN",msgID,myID,myID,*,-1,T1,0,0, Tr);
74. Else if myRef == Null AND I'm not clusterhead then
75.   RTSP("ERN", msgID, myID, myID, myCH,
76.        -1, T1, 0, 0, Tr);
77. End if
78. // if no msg received for T, start contest for reference
79. If no ERN msg received for T AND I'm clusterhead
80.      AND myID!=myRef AND myEnergy>low then
81.   // contest for reference node by broadcasting ERN
82.   myRef = myID;
83.   RTSP("ERN", msgID, myID, myID, *, myID, 0,
84.        0, 0, Tr);
85. End if
86. //a reference node broadcasts every after T period
87. If myID == myRef AND didn't broadcast for T AND
88.      myEnergy > low then
89.   // an active reference node rebroadcasts after T
90.   RTSP("ERN", msgID, myID, myID, *, myID, 0,
91.        0, 0, Tr);
92. End if
93. // if out of synchronization, send sync.request
94. If ((new REP msg overheard OR ERN msg received)
95.      AND myRef == msg→refNodeID AND
96.       local time much different than msg→Tr) OR
97.       skew is much lesser or greater than 1 then
98.    // node is out of synchronization; so send request
99.    RTSP("REQ", msgID, myID, myID, myRef,
100.        myRef, T1, 0, 0, Tr);
101. End if
102. //estimate offset & drift using two data points
103. If two data points, with one new, are available then
104.   Calculate α and β using the last two data points;
105.   Update the local clock to become synchronized;
106. End if
```

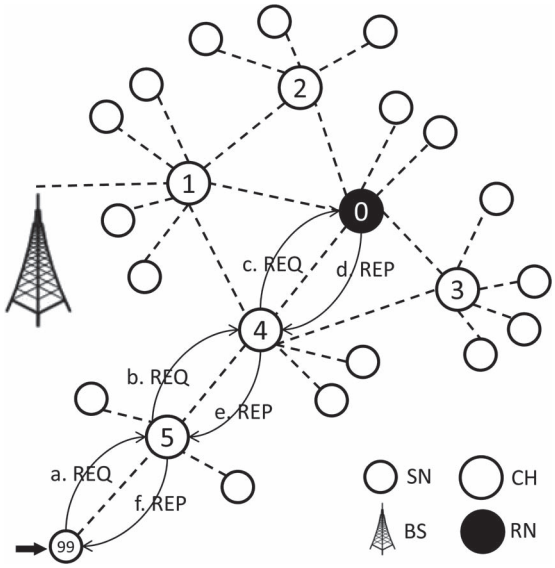

Fig. 5. Example of request-and-reply mechanism in RTSP algorithm. The request is recursed only when intermediate nodes are not synchronized.

Fig. 5 explains the request-and-reply mechanism of RTSP algorithm with the help of an example clustered network.

In Fig. 5, node 99 initiates an RTSP-REQ at its local time T1 which is received by an intermediate node 5 at its local time T2. The node 5 stores the identification of node 99 in myClient, values of T1 and T2 in T1old and T2old, respectively, and then forwards this request at its local time T1 to node 4 which recursively forwards it to the reference node 0. On receiving the request at its local time T2, the reference node 0 processes the request and sends back an RTSP-REP message at its local time T3. When node 4 receives the RTSP-REP reply at time T4, it performs the following six actions: 1) calculates the value of d, 2) adjusts the timestamps by adding d to Tr, 3) records the global and local timestamps, 4) retrieves the values of T1 and T2, 5) calculates the value of new Tr by adding to it the elapsed time since T4, and 6) forwards the reply to node 5 at its local time T3. Node 5 also performs the six actions and then recursively forward the reply to the node 99. Similarly, node 99 performs the first three actions only since it is not an intermediate node. Note that a reply follows exactly the same path as request, but a new request/reply message may follow a different path from the previous message.

In the next few sections, we describe some unique features of the RTSP algorithm which enhance its performance.

C. Adaptive Re-Synchronization Interval

A simple linear model for clock, as given in (4), is not reasonable for long time intervals because of the drastic changes in clock drift due to any sudden change in the environmental temperature, humidity, etc. That is why clock drifts are repeatedly reassessed by re-synchronizing the nodes periodically at a static interval. However, a static period is not feasible for dynamic environments. Therefore, we use adaptive re-synchronization interval in the RTSP algorithm, which allows each individual node to decide if and when to initiate a time synchronization request based on either the value of skew or the deviation of local time from the global time as follows (Algorithm 1: RTSP, line 93–101).

1) ***When skew is far away from 1:*** A node is considered to be perfectly synchronized when the current value of skew or drift is 1. Therefore, the value of skew can tell a node if it should initiate a time synchronization request or not. Accordingly, if the current value of skew is much lesser or greater than 1, the node will request timing information by sending an RTSP-REQ message to the reference node; otherwise, it will not request any update.
2) ***When local time is different from global:*** Although an overheard message is not yet compensated for offset, skew, or propagation delay; however, it can provide a rough idea of the global time. On overhearing an RTSP-REP message, a node can check if its local time is almost same as the global timestamp in Tr field. If the two times are much different, depending on the required accuracy, the node will initiate a time synchronization request; otherwise, it will not request any update.

Such an adaptive re-synchronization interval helps reduce the overhead of unnecessary time synchronization requests in the long run. It also provides for the long-term synchronization in a self-configuring way. In addition to this, the RTSP algorithm (Algorithm 1) performs a few other functions, which include restarting the process for election of reference node on detecting a failure, broadcasting the RTSP-ERN messages periodically by an active reference node, and calculating the values of α and β for updating the local clock.

D. Aggregation of the Synchronization Requests

In order to further reduce the number of time synchronization requests and hence energy consumption, the idea of request aggregation is supported in the RTSP algorithm. Consider the example in Fig. 5 and assume that another node under clusterhead 5 also initiates an RTSP-REQ message soon after node 99. The intermediate node (i.e., node 5) forwards only the first request to the reference node, waits for a reply, and then forwards this reply to all clients. In order to aggregate synchronization requests, node 5 temporarily saves information on msgID, myClient, T1, and T2 for both requests. It then forwards only one request to node 4. On receiving a reply, it uses the saved information for forwarding the reply to both requesters.

E. Security

WSNs face many security challenges due to their environment and constraints, which include vulnerability to physical capture, the use of insecure and unreliable wireless channel, and availability of the limited resources such as computing, memory and energy [18]. Moreover, the traditional protocols for time synchronization in WSNs do not consider security and hence prone to several kind of security attacks [19]. Although security is not the prime objective of RTSP, it provides some basic level of protection against the attacks on time synchronization in which a single compromised node may propagate incorrect timing information through the network. Simple techniques such

as redundancy, authentication, and refusal to forward corrupt synchronization information are used to tackle the attacks on time synchronization as follows.

1) ***Redundancy:*** RTSP algorithm provides some level of redundancy to avoid dependence on just one neighbor. For example, instead of receiving timing information always from the parent node like FTSP, each new request/reply message in RTSP may follow a different path. Also, a node may receive broadcasts by the reference node via a different neighbor at a different time. This provides redundancy for basic level of security.
2) ***Authentication:*** A node that wants to join the network is authenticated first by using any light-weight authentication protocol [18], [20]. Moreover, the RTSP algorithm is run only when message is received from an authenticated node (Algorithm 1: RTSP, line 1–3).
3) ***Discarding the corrupt information:*** Each RTSP message carries the current or real time of the sender in Tr field, which helps identify any timing information that is very much deviating from the recent data. A node can discard any corrupt message in order to stop the propagation of incorrect timing information through the network.

F. Energy Awareness and Efficiency

Nodes with different residual energy, i.e., low, medium, and high may exist in a WSN at any time. However, low-energy nodes should not contest for the reference node. Moreover, when an active reference node becomes low in energy, it should retreat to an ordinary node, and the process for election of the reference node should take place again. The RTSP algorithm ensures that any node contesting for reference node has sufficient residual energy (Algorithm 1: RTSP, line 19–24, and 78–85) so that it can stay active for a long time, broadcast RTSP-ERN messages regularly (Algorithm 1: RTSP, line 86–92), and actively reply to the queries forwarded to it. It will also save on the energy consumed by frequent re-elections of the reference node otherwise. However, when the reference node becomes low in residual energy, it retreats to an ordinary node, and the election process takes place again (Algorithm 1: RTSP, line 19–24).

In RTSP, the frequency of broadcasts by the reference node is 1/10th of the frequency in FTSP which saves huge amount of energy. Moreover, adaptive nature of the algorithm reduces unnecessary time synchronization requests and hence saves on energy. Note that once a node becomes synchronized, it can reply to time synchronization requests routed through it without any need to forward them to the reference node. This reduces burden on the reference node and saves on energy by not forwarding the request all the way to reference node.

IV. Error Analysis of RTSP

The uncertainties in massage delivery include the following time delays: send, access, transmission, propagation, reception, receive, interrupt handling, encoding, decoding, and byte alignment time [5], [6], [16]. Although the RTSP algorithm uses request-and-reply mechanism similar to TPSN but in a recursive manner, it uses MAC-layer time-stamping of messages like FTSP that removes all the sources of uncertainties in message delivery delays except the propagation delay [6]. Therefore, we can follow the TPSN-like approach in order to analyze the errors of RTSP algorithm.

Consider the request-and-reply mechanism of RTSP in Fig. 2 where node A sends a request at T1 to another node B that receives it at time T2 according to their local clocks. However, the real time measured by an ideal clock at these nodes is denoted by t1 and t2, respectively. The following equation was derived for TPSN in [5]:

$$t_2 = t_1 + S_A + P_{A->B} + R_B \tag{8}$$

where $\mathrm{S_A}$ is the time taken to send message (send + access + transmission time) at node A, $\mathrm{R_B}$ is the time taken to receive the message (reception + receive time) at node B, and $\mathrm{P_{A->B}}$ is the propagation time between nodes A and B.

As the MAC-layer time-stamping of messages removes all the sources of uncertainties except the propagation delay [6], (8) can be written as follows, for RTSP:

$$t_2 = t_1 + P_{A->B}. \tag{9}$$

The corresponding equation in terms of local clocks, which involves clock drifts as well, can be written as follows:

$$T_2 = T_1 + P_{A->B} + D_{t1}^{A->B}. \tag{10}$$

Now, the relative drift between the two nodes from t1 to t4, i.e., $RD_{t1->t4}^{A->B}$, is the difference of their respective drifts, which can be positive or negative depending on the value of two drifts, and is given by the following equation:

$$RD_{t1->t4}^{A->B} = D_{t1}^{A->B} - D_{t4}^{A->B}. \tag{11}$$

From (11), we can get the value of $D_{t1}^{A->B}$ and put in (10) to get the following equation:

$$T_2 = T_1 + P_{A->B} + D_{t4}^{A->B} + RD_{t1->t4}^{A->B}. \tag{12}$$

After receiving the request message at time T2, node B forwards this request to another node in a recursive manner until it reaches the node N, which replies with timing information that is forwarded to node B through the same path. The node B at time T3 forwards this reply to node A which receives it at time T4 according to their local clocks. Using the analysis similar to (10), following equation can be derived:

$$T_4 = T_3 + P_{B->A} + D_{t3}^{B->A}. \tag{13}$$

Note that $D_{t3}^{B->A} \approx D_{t4}^{B->A}$ and $D_{t4}^{B->A} = -D_{t4}^{A->B}$. Hence, (13) can be written as follows:

$$T_4 = T_3 + P_{B->A} - D_{t4}^{A->B}. \tag{14}$$

Subtracting (14) from (12) gives the following equation:

$$T_2 - T_4 = T_1 + P_{A->B} + D_{t4}^{A->B} + RD_{t1->t4}^{A->B} - T_3 - P_{B->A} + D_{t4}^{A->B}. \tag{15}$$

Re-arranging (15), and substituting $(P_{A->B} - P_{B->A})$ with P^{UC} gives (16), where P^{UC} represents the uncertainty in propagation time

$$(T_2-T_1)-(T_4-T_3)=P^{UC}+RD_{t1->t4}^{A->B}+2D_{t4}^{A->B}. \quad (16)$$

As we know that the offset $\Delta = ((T_2 - T_1) - (T_4 - T_3))/2$, therefore (16) can be written as follows:

$$2\Delta = P^{UC} + RD_{t1->t4}^{A->B} + 2D_{t4}^{A->B}. \quad (17)$$

To correct the clock at T4 at node A, subtract the value of $D_{t4}^{A->B}$ from Δ, which results in the following equation:

$$Error = \Delta - D_{t4}^{A->B} = \frac{P^{UC}}{2} + \frac{RD_{t1->t4}^{A->B}}{2}. \quad (18)$$

From (18), we can conclude that the RTSP algorithm has only two sources of error:

1) ***Variation in Propagation Delays*** $(\boldsymbol{P^{UC}})$: The RTSP algorithm assumes that the propagation delay in one direction is exactly equal to the other. When this is true, one source of synchronization error is eliminated and accuracy of the time synchronization is improved further. However, this assumption may not be true in wireless communication. Consequently, each hop might add possibly an inaccurate value of d to the reference time Tr in order to find the new value of Tr (Algorithm 1: RTSP, line 56). Fortunately, the variation in propagation delays can be neglected. Since propagation delay is calculated as distance between two nodes divided by the speed of radio signal through air that is close to speed of light, the variation in propagation delays can be neglected because the change in two determining factors is negligible during the short interval between T1 and T4. For example, the maximum increase in distance between two nodes moving at the speed of 1000 km/h will be only 0.0006 m/μs. Similarly, the speed of radio signal affected by changes in the air temperature and humidity is negligible for a distance of hundreds of miles; rather totally negligible for a distance of few hundred meters in WSNs. Therefore, we can conclude that the variation in propagation delays in WSNs is usually negligible.
2) ***Relative Drift Between Local Clocks*** $(RD_{t1->t4}^{A->B})$***:*** The hardware clock in a low-cost sensor node is usually based on quartz crystal oscillator that is vulnerable to huge drift from the ideal clock. However, (18) shows that the RTSP is sensitive only to the "relative drift" between the two clocks instead of the drift from ideal clock. It calculates the relative drift (β) and offset (α) for (4) using 2LR as follows in (19)–(20), where x_i is the global time while y_i is the local time:

$$\beta = \frac{x1 - x2}{y1 - y2}. \quad (19)$$

$$\alpha = \overline{y} - \beta\overline{x}. \quad (20)$$

The skew or drift between two clocks can be compensated using (4), (19) and (20). Consequently, the effect of this source of error is also minimized by the RTSP algorithm.

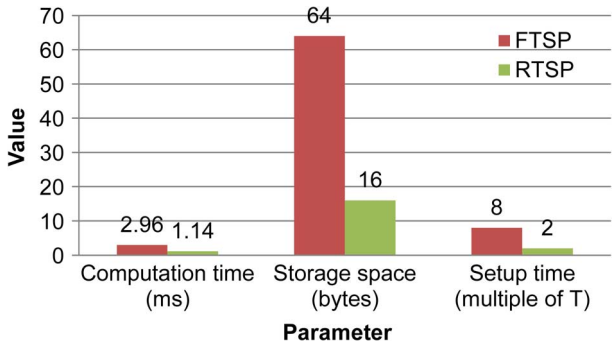

Fig. 6. Computation and memory requirements of RTSP versus FTSP.

A. Propagation of Errors in Multi-hop

Any error at one hop may increase when it propagates to the next hops. However, in case of time synchronization, the error may not always increase due to different sign and magnitude of the drift rate of sensors.

Let, in a network of k hops, every hop may experience a random error of δ. As the sum of randomly distributed errors is the square root of the sum of the squares of errors on each hop, the total error at kth hop may be calculates as

$$Error_{k-hop} = \sqrt{(\delta_1)^2 + (\delta_2)^2 + \cdots + (\delta_k)^2} \quad (21)$$

which can be written in a simpler form as follows:

$$Error_{k-hop} = \delta\sqrt{k}. \quad (22)$$

Therefore, the total error at kth hop may be in the order of $\delta\sqrt{k}$ which means that the accuracy of time synchronization does not decrease much with the increase in number of hops.

B. Bounds on Total Error in Multi-hop

According to (22), the total error at kth hop may be in the order of $\delta\sqrt{k}$. However, due to different sign and magnitude of error at each hop, the errors may sum up to a much smaller value which can be zero as well. Extreme situations occur when all the errors are either negative or positive. Accordingly, the total error at kth hop is $-\delta\sqrt{k}$ for all negative values of δ while $+\delta\sqrt{k}$ for all positive values of δ. This determines the bounds on total error in multi-hop network as

$$-\delta\sqrt{k} \leq Error_{k-hop} \leq +\delta\sqrt{k}. \quad (23)$$

V. Efficiency Analysis of RTSP

The RTSP algorithm is fast and efficient in terms of setup time or delay, computation and memory usage, and energy consumption. In this section, we analyze the efficiency of RTSP algorithm. Fig. 6 shows that the computation and memory requirements and setup delay of RTSP are much lesser than that of FTSP.

After the network starts up, RTSP can synchronize a node as soon as it receives two data points which can be requested by a node as soon as it knows the ID of reference node. Therefore,

TABLE II
SIMULATION PARAMETERS

Parameter	Description
No. of nodes	50, 100, 200, 500
Prob. of becoming clusterhead	5%
Topology	Random
Deployment area	50m x 50m to 500m x 500m
Mobility	High (pause time 0)
Mobility model	Random Walk
Processing delay (at each node)	1ms – 100ms
No of simulations (results averaged)	1000
Channel	Wireless
MAC	IEEE 802.15.4/ ZigBee
Antenna	Omni
Transmission range	30 m
Routing protocol	AODV

the setup time of RTSP is about 2T compared to 8T for FTSP, where T is the fixed interval for re-synchronization in FTSP.

The time-stamping mechanism of RTSP is much simpler when compared to FTSP. Moreover, RTSP calculates skew and offset using 2LR instead of 8LR in FTSP. Accordingly, RTSP requires much lesser time to execute its steps and lower memory to store the data points.

The RTSP algorithm consumes lesser energy compared to FTSP. It saves energy through infrequent reference broadcasts, and by reducing the number of requests through the adaptive re-synchronization interval and request-aggregation.

VI. PERFORMANCE EVALUATION

In order to evaluate the performance of proposed algorithm, we performed a simulation in MATLAB and collected data on accuracy or synchronization error and energy consumption.

A. Simulation Setup

A simulation was performed in MATLAB using the parameters given in Table II. Note the use of random topology, random walk mobility model, and pause time of zero, which makes the network highly dynamic. A time synchronization protocol that performs better in such a network is expected to perform even batter in a static environment. The main steps of the simulation are as follows.

1) A class for sensor node is defined with different attributes, such as ID, position (x, y, z), energy level, level in hierarchy, parent ID, current time, offset (α), and skew (β).
2) A set of network parameters are defined including the total number of nodes in the network, transmission range, deployment area, re-synchronization interval, mobility or pause time for each node, processing delay at each node, and number of simulation to run.
3) For total number of nodes, instances of the sensor node class are created, and random values are generated for the position of each node.
4) Communication links are defined according to the transmission range and distance between sensor nodes.

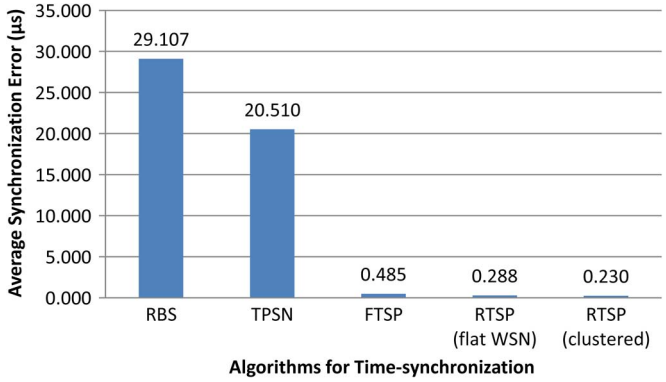

Fig. 7. Average absolute error of time synchronization.

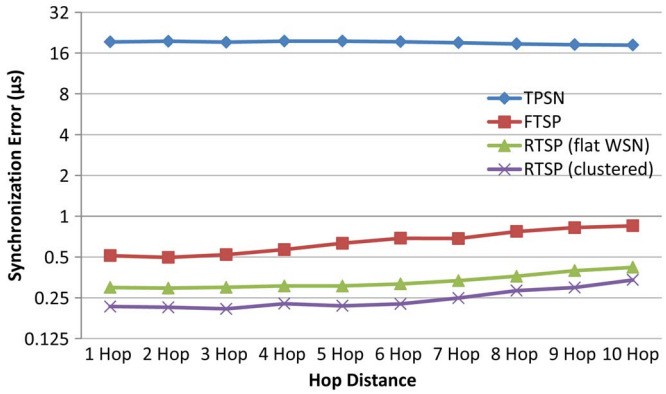

Fig. 8. Average absolute error of time synchronization (per-hop).

5) Four instances of the above network are created for the four protocols to run.
6) Network instances are customized for each protocol. For example, clusters are defined in the network instances for RBS and RTSP, and tree-like structure is defined in the network instances for TPSN and FTSP.
7) Protocols are run on their own instance of the network for several times, and results are stored in a workbook.
8) Incomplete results and outliers are filtered, and data is analyzed for synchronization error and energy consumption.
9) Results are plotted on graphs.

B. Results and Discussions

The simulation was run to collect data on average synchronization error, per-hop synchronization error, and energy consumption in the long run.

Fig. 7 shows the average absolute error of time synchronization for different algorithms, which is obtained by running the simulation for thousands of times with different parameters. The average absolute error of RTSP algorithm is 0.288 μs for flat network and 0.230 μs for clustered network, which is significantly lower than FTSP protocol that is 0.485 μs. However, the performance of RBS and TPSN is very bad compared to other algorithms.

As WSNs are usually multi-hop networks, it has always been interesting to compare the average synchronization error per-hop. Fig. 8 compares the average synchronization error of

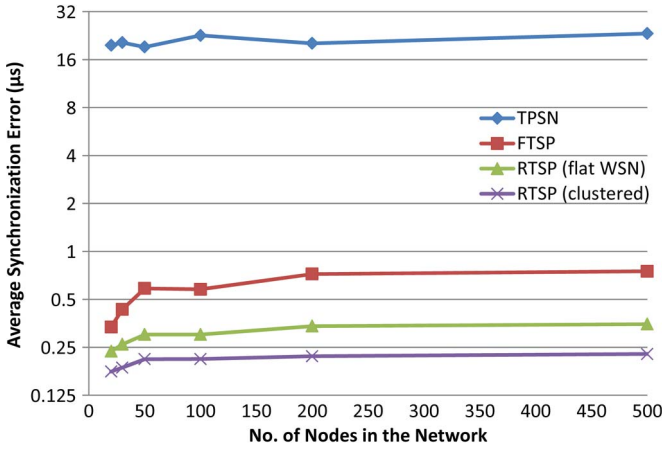

Fig. 9. No. of nodes versus average absolute error of time synchronization.

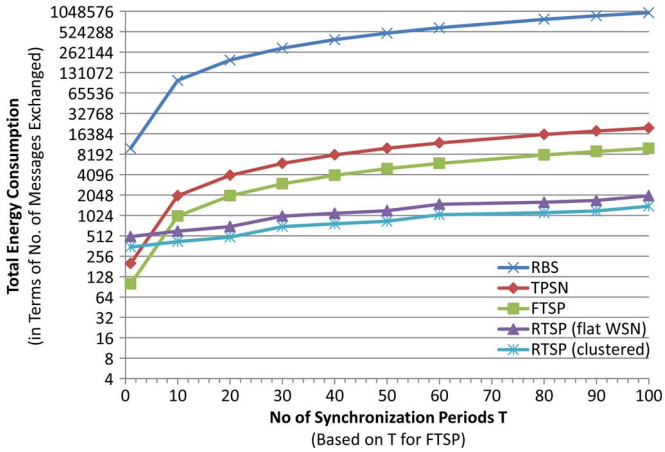

Fig. 10. Total energy consumption in the long run (for 100 nodes).

RTSP algorithm with TPSN and FTSP. We skip RBS from the comparison because it is not a multi-hop protocol. The Fig. 8 shows that average synchronization error of the RTSP algorithm for nodes at 10th hop is 0.420 μs for flat network and 0.388 μs for clustered network, which is even lesser than the average synchronization error of FTSP in a 1-hop network. Similarly, Fig. 9 compares the average synchronization error versus number of nodes in the network; again, the RTSP algorithm performs better than others.

Energy consumption is a very important factor in the performance of time synchronization algorithms. However, actual energy consumption many be affected by many factors, such as type of the hardware, software, and antenna. Therefore, we measure energy consumption in terms of total number of time synchronization messages exchanged among the nodes while assuming same type of hardware, software, and antenna. for all protocols. If the re-synchronization period is T seconds (based on FTSP) and N is the total number of nodes, then each node sends N messages in RBS protocol, two messages in TPSN protocol, one message in FTSP protocol [6], and only 0.14 messages in RTSP protocol in the long run. Calculating the total number of messages sent by RTSP is straight forward. On startup of the network, each node forwards one message, which is broadcasted by the reference node. Then, a node sends two RTSP-REQ messages and gets their replies accordingly. This makes five messages per node. However, once the nodes are synchronized, they need to send very less number of requests due to the adaptive behavior of RTSP algorithm. To find the total number of messages sent in RTSP in the long run, we collected data from simulation of a network of 100 nodes for very long time. Fig. 10 shows that, in the beginning, total energy consumption (i.e., no. of messages exchanged) of the RTSP algorithm is higher than TPSN and FTSP. However, it becomes equal to TPSN after four or five periods and equal to FTSP after seven or eight periods, and then remains always lower than the two protocols in the long run. Using the Curve Fitting Toolbox in MATLAB, we found the regression equation for RTSP curve which revealed that each node sends only 0.2 messages for flat network and 0.14 messages for clustered network in the long run, which means that it consumes 5 to 7 times lesser energy than FTSP.

TABLE III
COMPARISON OF RTSP WITH OTHER ALGORITHMS

Property	RBS	TPSN	FTSP	RTSP
Scheme used	Receiver-to-Receiver	Sender-to-Receiver	Sender-to-Receiver	Sender-to-Receiver
Time-stamping	App. layer	MAC	MAC	MAC
Sensitive to delays	propagation, decoding and interrupt handling	encoding, decoding and interrupt handling	propagation	propagation
Flooding	No	No	Frequent	Infrequent
Skew estimation	No	No	8LR	2LR
Compensation of propagation delay	No	Yes	No	Yes
Request-Reply	No	Iterative	No	Recursive
Adaptive	No	No	No	Yes
No. of messages (per node)	N	2	1	0.14 (in long run)
Energy usage	Very high	High	Low	Very Low
Accuracy (per-hop)	29.1µs (single-hop)	20µs	0.5µs	0.23µs

Table III provides a quick comparison of the RTSP algorithm with other global time synchronization algorithms.

VII. CONCLUSION

We have described the RTSP algorithm for global time synchronization in WSNs, which gives an average accuracy of 0.23 μs per hop in a large multi-hop clustered network using seven-times lesser energy than that of FTSP in the long run. An analysis of the sources of errors shows that the two sources of errors are variation in propagation delays and relative drift between local clocks, which are duly compensated by the algorithm. The accuracy of RTSP is improved by using the MAC-layer time-stamping based on SFD byte, which is simpler and more accurate. Further improvement in accuracy is gained by the compensation of propagation delay and adjustment of the timestamps at each hop. The RTSP algorithm achieves energy efficiency by using several techniques, which include the infrequent broadcasts by the reference node, skew-estimation using 2LR instead of 8LR, and reducing the number of time synchronization requests through the adaptive re-synchronization interval and aggregation of synchronization requests.

REFERENCES

[1] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Netw.*, vol. 7, no. 3, pp. 537–568, May 2009.

[2] D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[3] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. 1588-2008, 2008, pp. c1–269, (Revision of IEEE Std. 1588-2002).

[4] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Winter 2002.

[5] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. 1st Int. Conf. SenSys*, 2003, pp. 138–149.

[6] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. 2nd Int. Conf. SenSys*, New York, 2004, pp. 39–49.

[7] C. Dunn, D. Jefferson, S. Lichten, J. Thomas, Y. Vigue, and L. Young, "Time and position accuracy using codeless GPS," in *Proc. 25th PTTI Conf.*, Marina Del Rey, CA, 1993, pp. 169–179.

[8] R. Albu, Y. Labit, T. Gayraud, and P. Berthou, "An energy-efficient clock synchronization protocol for wireless sensor networks," in *Proc. WD, IFIP*, 2010, pp. 1–5.

[9] P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, "IEEE 1588-based synchronization system for a displacement sensor network," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 2, pp. 254–260, Feb. 2008.

[10] R. L. Scheiterer, C. Na, D. Obradovic, and G. Steindl, "Synchronization performance of the precision time protocol in industrial automation networks," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 6, pp. 1849–1857, Jun. 2009.

[11] T. Cooklev, J. C. Eidson, and A. Pakdaman, "An implementation of IEEE 1588 over IEEE 802.11b for synchronization of wireless local area network nodes," *IEEE Trans. Instrum. Meas.*, vol. 56, no. 5, pp. 1632–1639, Oct. 2007.

[12] D. Cox, E. Jovanov, and A. Milenkovic, "Time synchronization for ZigBee networks," in *Proc. 37th SSST*, 2005, pp. 135–138.

[13] M. Aoun, A. Schoofs, and P. van der Stok, "Efficient time synchronization for wireless sensor networks in an industrial setting," in *Proc. 6th ACM Conf. SenSys*, New York, NY, 2008, pp. 419–420.

[14] M. Akhlaq and T. R. Sheltami, "The recursive time synchronization protocol for wireless sensor networks," in *Proc. IEEE SAS*, Brescia, Italy, 2012, pp. 1–6.

[15] A. Hu and S. D. Servetto, "Asymptotically optimal time synchronization in dense sensor networks," in *Proc. 2nd ACM Int. Conf. Wireless Sens. Netw. Appl.*, 2003, pp. 1–10.

[16] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Trans. Comput.*, vol. C36, no. 8, pp. 933–940, Aug. 1987.

[17] *IEEE Standard for Information Technology—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std. 802.15.4-2003, 2003, pp. 1–670.

[18] Y. Wang, G. Attebury, and B. Ramamurthy, "A survey of security issues in wireless sensor networks," *IEEE Commun. Surv. Tutor.*, vol. 8, no. 2, pp. 2–23, 2nd Quarter, 2006.

[19] M. Manzo, T. Roosta, and S. Sastry, "Time synchronization attacks in sensor networks," in *Proc. 3rd ACM Workshop Security Ad Hoc Sens. Netw.*, 2005, pp. 107–116.

[20] V. C. Giruka, M. Singhal, J. Royalty, and S. Varanasi, "Security in wireless sensor networks," *Wirel. Commun. Mobile Comput.*, vol. 8, no. 1, pp. 1–24, Jan. 2008.

Muhammad Akhlaq (M'12) received the M.Sc. and M.Phil. degrees from University of the Punjab (PU), Lahore, Pakistan, in 2001 and 2005, respectively. He has also received Master of Ubiquitous Computing from Blekinge Institute of Technology (BTH), Karlskrona, Sweden in 2010. Recently, he has completed the Ph.D. degree in computer science and engineering from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in April 2012.

He has more than 10 years of teaching and research experience in different organizations, and more than 17 publications in well-known journals and conferences. He is coauthor of the Recursive Time Synchronization Protocol for WSNs.

Currently, he is working at KFUPM. His research interests include wireless ad hoc and sensor networks, ubiquitous computing, and context-aware computing

Tarek R. Sheltami (M'00) received the Ph.D. degree in electrical and computer engineering from the Electrical and Computer Engineering Department at Queen's University, Kingston, ON, Canada, in April 2003.

Currently, he is working as an Associate Professor at the Computer Engineering Department at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. Previously, he was a Research Associate Professor at the School of Information Technology and Engineering, University of Ottawa, Ontario, Canada. He has worked at GamaEng Inc. as a Consultant on Wireless Networks (2002–2004). He has also worked in several joint projects with Nortel Network Corporation. He was one of the two project managers on "A Testbed Wireless Ad hoc Network."

Dr. Sheltami is the coauthor of the Warning Energy Aware Clusterhead infrastructure protocol and the Virtual Base Station On-demand routing protocol. He has been a Member of the technical program and organizing committees of several international IEEE conferences.