Time Synchronization for Predictable and Secure Data Collection in Wireless Sensor Networks

Shujuan Chen, Adam Dunkels, Fredrik Österlind, Thiemo Voigt
Swedish Institute of Computer Science
{shujuan,adam,fros,thiemo}@sics.se

Mikael Johansson
KTH Stockholm
mikaelj@s3.kth.se

***Abstract*— Wireless sensor networks are trying to find their way from relatively undemanding applications such as environmental monitoring to applications such as industrial control, which have stronger requirements in terms of security and predictability. Predictability cannot be achieved without coordination and the coordination of distributed entities and events requires time synchronization. Towards this end, we present a secure time synchronization service, that as our experimental results show does not degrade time synchronization accuracy. Based on the time synchronization service we implement time slotted data collection and present results that show that this way we can provide a predictable data collection service.**

I. Introduction

Wireless sensor networks consist of small devices with sensing and processing facilities as well as a low-power radio interface. While previously, applications where mostly dedicated towards environmental monitoring and other surveillance tasks, we are now seeing the emergence of industrial control applications based on wireless sensor networks [2]. Industrial control applications are more demanding than environmental monitoring applications, with respect to predictability and security, for example when monitoring critical equipment.

Due to the distributed nature of wireless sensor networks, time synchronization is a prerequisite for predictability. Furthermore, time synchronization enables among further things ordered event logging as well as the possibility to deploy a TDMA schedule. Towards this end, we have designed and implemented a secure time synchronization service for the Contiki operating system [3]. As TPSN [7], our time synchronization protocol builds a tree structure that is used to perform pair-wise synchronization. The implementation of the security services utilizes the hardware-based AES security operations supported by the CC2420 radio.

Our main results and contributions are the design and implementation of a time synchronization protocol that has a low overhead in terms of message exchanges. Furthermore, experiments with real hardware confirm that the security service does not degrade the time synchronization accuracy. We present experimental results that show that a data collection protocol using slotted communication based on the network-wide time synchronization is able to achieve a predictable data collection service and low latency.

The rest of this paper is structured as follows: After background information in the next section, we present our protocol design and implementation in Section III. Experimental results are presented in Section V and are discussed in Section VI. We review related work in Section VII and conclude the paper in Section VIII.

II. Background

A. The Contiki Operating System

Contiki is an operating system for small embedded devices such as sensor nodes. It features an event-driven kernel and a blocking wait based on the protothread abstraction [4] to simplify application development. Contiki also supports loadable modules.

B. Telos Sky motes security

The hardware platform we have used is the Tmote Sky. Tmote Sky is a low power wireless module for use in low-power sensor network applications [9]. It is integrated with a TI MSP430F1611 microcontroller and a CC2420 [11] radio that is compliant with the 2.4 Ghz IEEE 802.15.4 specification. The CC2420 radio provides high-level security based on AES encryption using 128-bit keys. Security operations are performed within the transmit and receive FIFOs on a frame basis. The CC2420 can perform MAC security operations (encryption, decryption and authentication) on frames within the TXFIFO and RXFIFO. These operations are called inline security operations. To enable these inline security operations, we first need to configure the security control

registers. Then we set the keys for transmission and receiving. For counter-mode operations, we also need to set up the nonces (numbers used once) for each transmission and reception. After these settings, we can add the security operations in the CC2420 MAC driver for each transmission and reception [11].

C. Time synchronization protocols

According to the varying requirements, such as diverse precision requirements, network density, degree of mobility and topology stability, there exist several time synchronization protocols. Time synchronization is typically based on message exchange containing the timestamp and the measurement of delay. There are three basic solutions for synchronization between two nodes:

1) sender/receiver-based synchronization
2) receiver/receiver-based synchronization
3) delay measurement synchronization

One-way message exchange is commonly used in receiver/receiver synchronization solutions such as in the Reference Broadcast Synchronization (RBS) [5], while a two-way message exchange is typically used in sender/receiver-based synchronization solutions, such as the Timing-sync Protocol for Sensor Networks (TPSN) [7]. There are also some synchronization protocols based on one-way message exchange as well as the measurement of delay. An example of such a protocol is Delay Measurement Time Synchronization (DMTS) [10].

III. Design and Implementation

In this section we present the design and implementation of our time synchronization protocol and the predictable data collection scheme.

A. Time synchronization

According to Sundararaman et al. [1], TPSN achieves a good compromise between communication overhead, synchronization precision and computational complexity. DMTS has the advantage of low computational complexity and extremely low communication, but lower synchronization precision compared to TPSN due to more uncertainty from delay factors. RBS obtains high synchronization precision at the expense of high communication overhead, computational complexity, and the disadvantage that reference nodes are not synchronized.

Based on Sundararaman's comparison, we take the idea of TPSN [7] to achieve the good trade-off among the performance metrics. Using implicit topology formation in our protocol design instead of the two-phase approach taken by TPSN that explicitly establishes a hierarchical tree in the first phase, we expect that our protocol achieves lower computational requirements and better mobility support than TPSN while maintaining the same performance in other metrics. The main features of our solution are:

a) Pair-wise time synchronization: Figure 1 shows the sender/receiver-based pairwise synchronization. Let $T0$ be the client node (sender) timestamp on the request message, $T1$ the parent node (receiver) timestamp at arrival, $T2$ the parent timestamp on departure of the reply message and $T3$ the client timestamp at arrival.

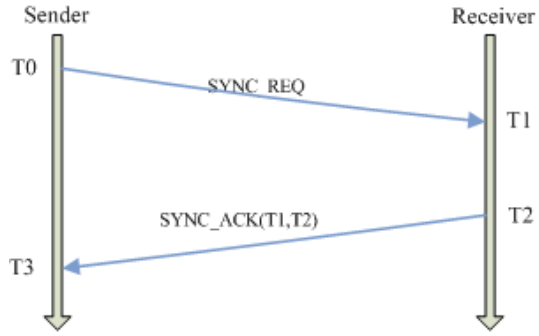

Fig. 1. Sender/receiver-based pairwise synchronization

Assuming that both one-way delays are equal, we can calculate the clock offset and the roundtrip delays as

$$offset = [(T1 - T0) + (T2 - T3)]/2,$$
$$delay = (T3 - T0) - (T2 - T1).$$

Knowing all the timestamps, the client (sender) decrements its logical clock by the estimated offset to achieve synchronization with the parent node. The delay information is necessary to prevent certain attacks as described in Section IV.

b) Application-level time synchronization: For our implementation we have decided to deploy application-level time synchronization with time stamping at the MAC layer. By offering an application-level time synchronization service we can reduce the degree of coupling between service and MAC layers: the service is not based on the specific MAC protocol but on the interface at the application level. Meanwhile, we can still timestamp the packet at the MAC layer to reduce uncertainty from varying software stack processing time.

c) Network-wide time synchronization: To achieve network wide time synchronization, we implicitly establish a hierarchical tree over the network with the root node as time reference node. Then we perform pairwise time synchronization along the tree as shown in Figure 2.

To form a hierarchical tree, all sync packets contain a hopcount. When a node receives a sync message, it checks the hopcount of the sender. If its own hopcount is greater than the sender's hopcount plus one, it updates its hopcount and sets its parentID to be this sender's ID. In this way, the hierarchical tree can be constructed implicitly and adapt dynamically to topology changes.

We assume that only the sink node has the right to initialize a network-wide synchronization. Its network synchronization message NETSYNC_BROADCAST triggers the nodes to start a new round of synchronization. The sink node sends the message NETSYNC_BROADCAST periodically to achieve a periodical resynchronization of the network. We need to perform periodical synchronization because the clock might lose synchronization due to crystal frequency difference or clock drift caused by environmental conditions such as the temperature.

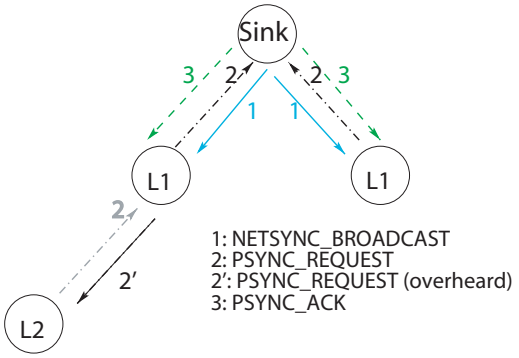

Fig. 2. network-wide time synchronization

When nodes at level 1 get the NETSYNC_BROADCAST message from the sink node, they update their hopcount and parentID, wait a random delay (to avoid collisions with other nodes in the same level), and then send the PSYNC_REQ to the sink node to start the pairwise synchronization. Nodes in lower levels overhear the PSYNC_REQ from their parents, wait an estimated roundtrip delay (to allow the parent to finish synchronizing) plus a random delay (to reduce collisions with nodes on the same level) and then trigger time synchronization by sending a PSYNC_REQ.

When receiving a PSYNC_REQ message addressed to it, a node immediately returns a PSYNC_ACK message to the requesting node. The timestamp of sending and receiving a PSYNC_REQ is included in the PSYNC_ACK message and send back to the requesting node.

When receiving a PSYNC_ACK message, the node records the receiving and sending timestamp of the PSYNC_ACK message. At that point, the node has the four required timestamps and can synchronize its clock to its parent as described above.

B. Predictable data collection with low latency

For control applications data collection with predictable latency is of crucial importance. In this section we present a multi-hop low latency data collection protocol based on Treeroute, one of the convergecast protocols implemented in Contiki.

The treeroute routing protocol is based on Trickle [8], a flooding protocol, originally intended for code propagation in sensor networks. Once one node is set as the sink, it sends the trickle routing broadcast with hopcount 0. When a neighboring node receives this broadcast it checks if its hopcount is equal to one. If not, it updates its neighbor table and changes its hopcount to one. Then it waits a random delay and send a new routing broadcast with its hopcount to inform the neighboring node in next level to update their hopcount and neighbor table. This process continues until all the nodes have updated their hopcount and neighbor table.

At the same time, a node might also get the broadcast from the neighboring nodes in the same level or from the children in the next level, it might also add these nodes into its neighbor table if the neighbor table has an neighbor entry with higher hopcount or same hopcount but with lower signal strength. After this routing process, every node has a hopcount, representing its level in the network, and a table of all its neighbors with good connectivity.

Sensed data is forwarded from the sensor node to the sink along the routing path derived from the neighbor table in each node. Data packets are forwarded to the best neighbor, i.e. the neighbor node with highest signal strength of the ones that have the lowest hopcount. To maintain reliable communications, retransmissions with a maximum limit are carried out if an implicit ACK is not recorded within a fixed period. An implicit ACK is when the sender overhears that the receiver forwards the sent packet, and is taken as an acknowledgement that the packet has been received successfully.

To achieve low-latency and reliable data collection, we must reduce collisions, thus reducing contention time and the number of retransmissions. Based on our time synchronization service, we deploy a TDMA scheme assigning different slots to each node. A sensor node sends data only in its slot and thus avoids collision with other nodes. In our data collection protocol, slot assignment is done according to a time schedule constructed by

dividing one second into N slots and assigning one slot to each node according to its (unique) $nodeid$. In this way, data collection becomes reliable and fast.

IV. Security Goals and Analysis

In this section we present the main goals for the secure time synchronization and data collection services, discuss attack scenarios and implemented countermeasures.

A. Secure time synchronization

We have two main goals for the secure time synchronization service: First the protection of the critical data (i.e. the timestamps) from being modified and delayed. This is necessary to get the correct time offset for the clock correction. Second, we must prevent nodes from synchronizing to a malicious node in order to guarantee that sensor nodes synchronize only to the legal nodes in the network. Next, we present a number of possible attack scenarios and discuss to what extend our secure service provides effective countermeasures.

Modification of timestamps

One possible attack scenario is when an adversary attempts to modify the values of the timestamps in sync packets. This can be counteracted using cryptographic methods, such as data authentication or encryption. To prevent the modification, the timestamps in a sync packet are authenticated using CBC-MAC (Cipher Block Chaining Message Authentication Code) for authentication, hence there is no way to change the timestamps without being detected. A secret key is shared between the sender and receiver for the authentication.

Pulse-delay attack

As described in [6], the pulse-delay attack can be performed by jamming the initial pulse, storing it in memory and then replaying it later at a arbitrary time. By manipulating the pulse-delay, an adversary can arbitrarily change the computed time offset. Any delay attacks which lead to delayed timestamping can be detected by setting a threshold value according to the estimated maximum one-way transmission delay. A one-way transmission delay includes transmission processing time, media access time, propagation delay, and receiving processing time. A node calculates the round-trip delay in a pairwise synchronization. If the delay exceeds the threshold value, it discards the packet.

Replay attack

A replay attack is when an adversary captures a packet and replays it at a later time. The old packet with an old timestamp leads to a false offset and could compromise the time synchronization process. Replay attacks can be detected by the authenticated timestamps or via sequence numbers. A node maintains the value of last timestamp or sequence number from its parent. When it receives a broadcast from its parent, it checks if it is an old message. If so, it regards it as a replay message and simply discard it.

Compromised nodes

Compromised node attacks are common for wireless sensor networks without physical security and could allow an adversary to manipulate the compromised node to send false data. Resisting attacks from compromised nodes goes beyond the scope of cryptographic methods and is currently not considered in our design.

B. Security for low-latency data collection

The main design goals for a secure low-latency protocol are the following: First, the data should be protected from being modified and disclosed by malicious attackers. Second, we want to make sure the data packets arrive at the destination node, i.e. the sink, with tolerable latency. Finally, we want to ensure that data packets are not the old ones replayed by an adversary.

An adversary can easily modify the data if no authentication mechanism is used. If the data is confidential, it needs to be encrypted to prevent disclosure. To protect the data in transit, we propose to use the security mode CCM to encrypt and authenticate all data packets based on a secret key shared by the sink and all sensor nodes.

The latency for data collection using Treeroute depends on the hopcount of sender, i.e. the distance to the sink, as well as the number of retransmissions. An adversary might arbitrarily delay the data packets, but the retransmission mechanism according to the implicit ACK in Treeroute will, to some extent, counter this attack. If the sender does not receive the implicit ACK in a short period, it regards it to be lost and retransmit it. After a maximum number of retransmissions, a node removes this neighbor and finds another forwarding node.

Replay attacks can also happen. If the data packet contains data about a time-critical event, it should not be reported repeatedly to trigger a false alarm. So sequence numbers can be used in each packet for each sensor node. The sink node needs to keep track of the last sequence number of data from each node. When the replayed packet is received, the sink node discards it. As an alternative, the authenticated timestamp can also be used to ensure data freshness.

V. Experimental Evaluation

In this section we evaluate our protocols by conducting experiments on real hardware using the Contiki operating

system. For all experiments we set the tick frequency to 512, i.e. one tick equals 1/512 second $\approx$ 2 milliseconds.

A. Time Synchronization Accuracy

1) Synchronization without security: In this part, we disable the security operations for all communications, so the time synchronization is not protected. Eleven nodes are deployed for the experiments, respectively, running in a single-hop fashion and a multi-hop fashion.

a) Single-hop networks: In a single-hop network, one node acts as the sink node and initiates the network-wide time synchronization. The remaining ten nodes synchronize themselves to the sink node. The topology is shown in Figure 3.

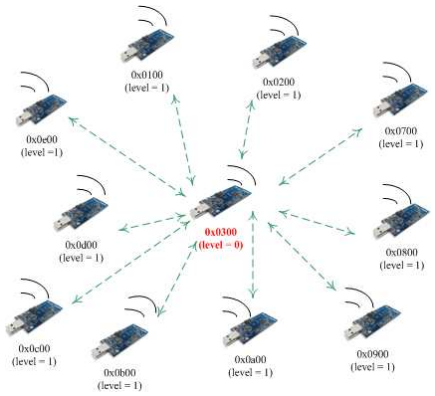

Fig. 3. Single-hop network topology

The main problem here is that all neighboring nodes need to send the requests (PSYNC_REQ) at different times in order to avoid collisions after receiving the NETSYNC_BROADCAST message from the sink at the same time. So a random delay is used for the neighboring nodes to send the request at a different time. We set the maximum random delay as: $random\_delay = max\_num * round\_delay * fa$, where fa a factor that adjusts the collision probability. In this experiment, let max_num to be 10, $round\_delay$ to be 6, and fa to be 10. Thus the maximum random delay is 600 (ticks).

All nodes run the time synchronization process and one node as acts the sink, periodically initiating time synchronization by sending the NETSYNC_BROADCAST message. In our experiment, we record the results for 50 periodical synchronizations for the sink node and three neighboring nodes. The results shown in Figure 4 shows that the three nodes achieve synchronization with an average synchronization error of 1.4 ticks. Meanwhile, the sink node receives about 97.6% of requests from the neighboring nodes. That is, only about 2.4% of the requests are lost due to collisions.

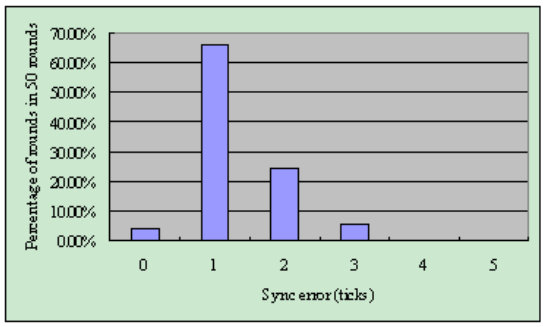

Fig. 4. Synchronization in a single-hop network without security

b) Multi-hop networks: To evaluate the synchronization performance in a multi-hop network, we place twelve sensor nodes in a configuration where three monitored nodes are three hops away from the sink, see Figure 5. We record the results for three nodes in level 3 for 50 rounds of periodical synchronization and analyse the synchronization error between these three nodes and their parent node. The results are shown in Figure 6.

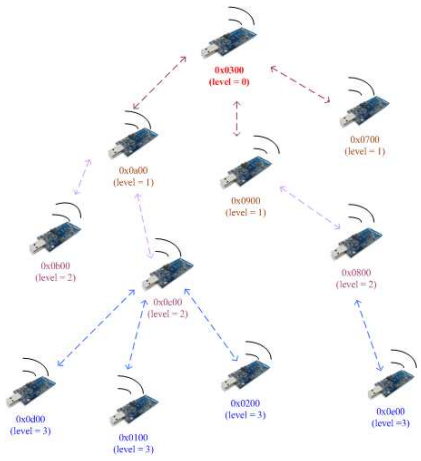

Fig. 5. Multi-hop network topology

The results in the figure show that the three nodes achieve synchronization with an average synchronization error of 1.1 ticks. The average synchronization error is lower than that of the previous test in a single-hop network. This is reasonable since only three nodes in the same neighborhood synchronize to the same parent. In contrast, in the single-hop network, ten neighboring nodes are synchronizing with their parent, the sink node. Thus the chance of collision between the neighboring nodes is much higher and leads to more uncertainty in the time-critical path for the single-hop experiment.

2) Synchronization with security: In this part, we enable the MAC-layer security operations to protect

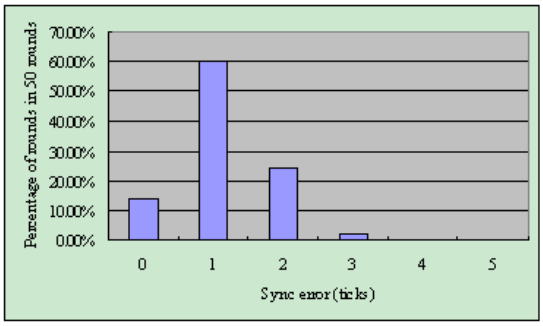

Fig. 6. Synchronization in a multi-hop network without security

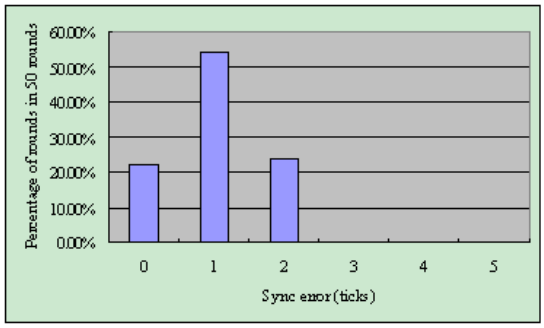

Fig. 8. Synchronization in a multi-hop network with security

the time synchronization process. The same network topologies as above is used for the experiments. In the experiment, we choose the authentication mode, CBC-MAC with 128-bit MIC, to protect the timestamps for synchronization from illegal modification. We do not use the encryption mode because we are not concerned if the timestamps are captured by an adversary but only if the timestamps are changed in transit. The MAC-layer security operation is transparent to the protocol, but it can detect the modification of data packet and discard the packet if the packet does not pass the MIC check.

a) Single-hop networks: For a single-hop network running with security, we record 50 rounds of periodical synchronization for three nodes synchronizing with the sink node. The results are shown in Figure 7.

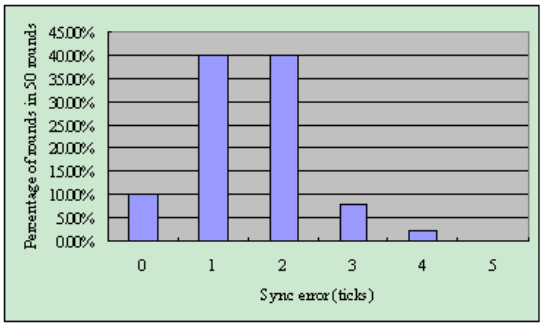

Fig. 7. Synchronization in a single-hop network with security

Figure 7 shows that the three nodes achieve synchronization with an average synchronization error of 1.5 ticks. Without security this error was 1.4 ticks.

b) Multi-hop networks : In the multi-hop scenario, we record 50 rounds of periodical synchronization for the three nodes synchronizing with its parent node. These three nodes are located in level 3 (i.e., 3 hops away from the sink node). The results are shown in Figure 8.

Figure 8 shows that the three nodes achieve synchronization to their parent with an average synchronization error of 1.0 ticks. Without security, the average error was 1.1 ticks. Also here, the difference to the results without security is 0.1 ticks which we consider to be a statistical error.

In the experiments, our time synchronization protocol achieves network-wide synchronization with synchronization error of about 1 to 2 ticks, i.e., 2 to 4 milliseconds. Moreover, synchronization with MAC-layer security works as well as synchronization without security. The time for security operation on a fixed length of synchronization packet should be deterministic. Hence, and as confirmed by our experiments, the synchronization error is not affected by the MAC-layer security operation.

B. Time-synchronized low-latency data collection

In this section, we conduct experiments to measure the end-to-end latency for data packets sent between sensor nodes and the sink node in the network. The network setup is shown in Figure 9.

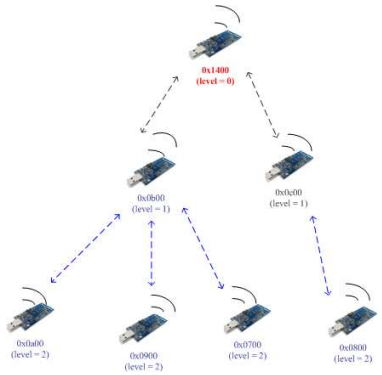

Fig. 9. Network for data collection experiment

Each node is assigned a unique slot during a slot period. Since each slot period is 32 ticks (around 64 ms), we can assign 16 slots per second.

We run a data collecting application based on time synchronization providing a global clock and Treeroute providing multi-hop routing. After the sensor nodes synchronize with the network, they send one data packet every second. The latency is measured for each data packet in ticks. In the sink node, we collect 5000 data packets from both one-hop nodes and two-hop nodes. Figure 10 shows the latency for packet delivery from one-hop nodes. The average end-to-end latency for one-hop data collection is 5.3 ticks (close to the single-hop communication latency for an isolated pair of nodes). Figure 11 shows the latency for data delivery from two-hop nodes. The average end-to-end latency for two-hop data collection is 10.3 ticks.

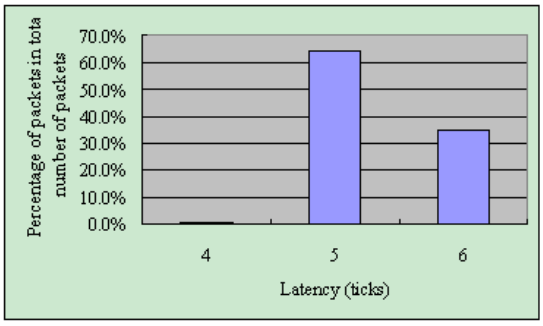

Fig. 10. Latency statistics in from one-hop nodes

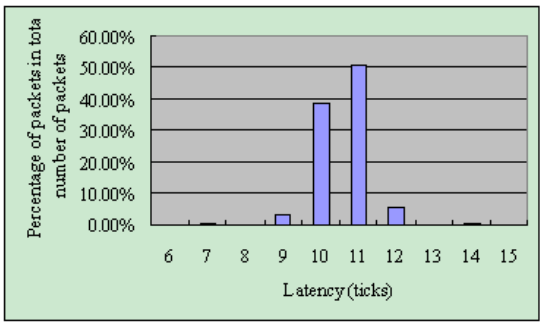

Fig. 11. Latency statistics in from two-hop nodes

The results show that the latency increases with the number of hops from the sink. The latency is stable so we can conclude that the sensor nodes are stably synchronized and collisions are eliminated by the schedule-based slotted data communication.

VI. Discussion

In this section we discuss problems caused by node failure or isolation. In a WSN, a node can die due to energy exhaustion or other interruption. Nodes can also be isolated from the network due to radio instability.

a) Node isolation: During our experiments we observed that a sensor node at level n occasionally overheard the timesync request from a node at level n-2, i.e. the parent's parent, and updated its hopcount to n-1. The problem is that this is just an occasional event caused by the fact that communication quality in wireless sensor networks can vary dramatically over time [12] and the node may never be able to overhear from that node again. Meanwhile the node ignores timesync request from node at the same level, n-1. In this case, the node will wait infinitely for the message from level n, but in fact it is too far to get timesync request again. As a result this node will be isolated from the time synchronization process.

To reduce the occurrence of this situation, we can set a threshold value of the signal strength of the timesync packets. Only if the signal strength of the received timesync packet is greater than the threshold value, the node will handle this packet and simply discard it otherwise. Another possibility is to define a timeout for each node. The timer will be reset every time a pairwise synchronization is done. If the timer times out, the node accepts timesync packet from other nodes and selects a new parent. This way an isolated node can get the chance to synchronize to the network with a new parent.

b) Node failure: This problem occurs when a node fails due to energy exhaustion or other unexpected interference. If the node is a leaf node this will not affect the time synchronization. But if the node is the parent of other nodes, then all its children will be isolated from the network. Also this problem can be solved using a timer. Each node checks if a timeout period has passed without any message from its parent node. If the timer expired, the node regards its parent as dead and updates its hopcount when getting message from other node. In this case, it might also get the message from nodes at the next level or the same level, but according to the rule, it will keep updating its hopcount until it gets the message from the neighbour node the highest level. In both cases, the value of the timeout period needs to be set properly according to the network synchronization period. It should be at least greater than one network synchronization period. In the current implementation, we set the time out period to be five times of network synchronization period.

VII. Related Work

In Reference Broadcast Synchronization (RBS) a reference node broadcasts a message and all the nodes in coverage will timestamp the local time of receiving

this reference message [1], [5]. Nodes then exchange the recorded times to gain knowledge of the time offset between themselves and other nodes in the network. RBS avoids the largest sources of error (send time and media access time) in the time-critical path and the synchronization error depends only on the difference in propagation time (often negligible), the time difference between receivers and the radio synchronization error, which is typically a few microseconds. In contrast to RBS, our protocol expects lower computational complexity and synchronizes the whole network including the reference node. In the time-critical path, if media access time and transmission time are deterministic, our protocol can expect an equivalent synchronization precision as in RBS. The precision difference to RBS will depend on the uncertainty of transmission time and media access time in the time-critical path.

The Timing-sync Protocol for Sensor Networks (TPSN) aims at providing network-wide time synchronization and establishing a unique global timescale by creating a self-configuring hierarchical tree. TPSN has two phases to achieve network-wide synchronization: the level discovery phase and the synchronization phase. In the level discovery phase a hierarchical tree is established. In the synchronization phase, pair-wise synchronization is performed along the edge of the tree. The synchronization accuracy does not degrade significantly with the number of nodes in the network, so the protocol is scalable, but due to its dependence on the hierarchical infrastructure, TPSN is not suitable for highly dynamic networks with mobile nodes [1], [7]. In contrast to TPSN, our protocol can expect lower communication overhead and better adaptability to topology changes through the dynamic implicit tree formation while maintaining an equivalent synchronization precision. The synchronization precision in our protocol depends on the uncertainty of the same factors as in TPSN, i.e., media access time, transmission time, propagation time and reception time in the time-critical path.

Delay Measurement Time Synchronization (DMTS) is very energy-efficient and computationally lightweight since it needs only one time broadcast to synchronize all the nodes in a single-hop network. The receivers measure the time delay and set their time as the sender's sending timestamp plus measured time transfer delay. The synchronization error will depend on how well the delay measurements are made along the time-critical path [1], [10]. Our protocol can expect higher synchronization precision since we eliminate some sources of synchronization error that exist in DMTS.

VIII. Conclusions and Future Work

In this paper we have presented a secure time synchronization protocol and demonstrated that the security mechanisms do not impact time synchronization accuracy. We show that our time synchronization protocol enabled the implementation of a predictable low-latency data collection service. Future work includes a quantitative performance analysis of the protocol and improved node scheduling algorithms to decrease data collection latency even further.

Acknowledgments

This work was partly financed by VINNOVA, the Swedish Agency for Innovation Systems, and the European Commission under contract IST-004536-RUNES.

References

[1] A.D. Kshemkalyani B. Sundararaman, U. Buy. Clock synchronization for wireless sensor networks: A survey. *Ad-hoc Networks*, May 2005.

[2] A. Bonivento, L.P. Carloni, and A.L. Sangiovanni-VIncentelli. Platform based design of wireless sensor networks for industrial applications. In *Design Automation and Test in Europe (DATE) 2006*, Munich, 2006.

[3] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[4] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys 2006*, Boulder, Colorado, USA, 2006.

[5] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[6] S. Ganeriwal, S. Capkun, C. Han, and M. Srivastava. Secure time synchronization service for sensor networks. In *WiSE*, September 2005.

[7] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, November 2003.

[8] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. NSDI'04*, March 2004.

[9] Moteiv. Tmote Sky). Web page. Visited 2006-12-07. `http://www.moteiv.com/products-tmotesky.php`

[10] S. Ping. Delay measurement time synchronization for wireless sensor networks. In *Intel Research*, June 2003.

[11] Chipcon product from Texas Instruments. CC2420 Datasheet). Web page. Visited 2006-12-07. `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.p`

[12] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *The First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, California, November 2003.