Outline

• Background of Wireless Sensor Networks
• Related Work
• TinySeRSync: Secure and Resilient Time Synchronization
• Conclusion and Future Work

Related Work

• NTP and GPS are not practical for sensor networks.
• Recent time synchronization techniques.
    • Receiver-Receiver based schemes.
      – Elson et al., ACM SIGOPS’02.
    • Sender-Receiver based schemes.
      – Ganeriwal et al., SenSys’03.
  – Global Time Synchronization.
    • Clock distribution schemes.
      – Munshi et al., SenSys’04.
    • Clock agreement schemes.
      – Li and Rux, INFOCOM’04.
Threats

- Single-hop Pair-wise Time Synchronization.
  - No message authentication.
    - Manzo et al., SASN’05; Ganeriwal et al., Wise’05
  - Time sensitive.
    - Jam and Replay attacks: Ganeriwal et al., Wise’05

- Global Time Synchronization.
  - Insider attacks

Our Contributions

- TinySeRSync:
  - Phase I
    - Secure single-hop pair-wise time synchronization
  - Phase II
    - Secure and resilient global time synchronization

Phase I:

Secure Single-hop Pair-wise Time Synchronization
Overview

• Goal:
  – Achieve time difference between two neighbor nodes in hostile environment.

• Existing work
  – Secure TPSN (Ganeriwal et al., Wise’05)
  – Problems with secure TPSN

• Our work:
  – Prediction-based MAC layer timestamp.
  – Hardware-assisted, authenticated MAC layer timestamp.

Secure TPSN

• Ganeriwal et al., Wise’05

• Estimate time difference and transmission delay.
  \[ \Delta = \frac{(t_2 - t_1)(t_3 - t_4)}{2} \]
  \[ d = \frac{(t_2 - t_3)}{2} \] is half of the transmission delay.

  – Security condition:
    • \( d < \) maximum expected delay.
    – \( K_{AB} \): secret key shared between A and B.
    – MIC: message integrity code

Problems in Secure TPSN

• Authenticated MAC layer timestamp.
  – MIC must be available when radio sends the MIC field.

• Software solution in Secure TPSN
  – Calculate MIC using TinySec (Karlof et al., SenSys’04)
  – Works for low data rate radio (e.g., 38.4 kbps in CC1000 used by MICA2).
  – Does not work for high data rate radio (e.g., 250kbps in CC2420 used by MICA2).
Prediction-based MAC Layer Timestamp

- **Sender side:**
  - When channel is clear, it adds sending time = current time + constant delay $\Delta$. $\Delta = 399.29$ us.
- **Receiver side:**
  - When the SFD field is received, it records the time as receiving time.

Hardware-assisted, Authenticated MAC Layer Timestamp

- **Hardware security support in CC2420**
  - Two modes
    - **stand-alone mode:** 128 bit AES encryption on message in stand-alone buffer.
    - **in-line mode:**
      - begin encryption when message is being sent out,
      - begin decryption after the whole message is received.
  - Using in-line mode, CC2420 can generate a 12-byte MIC on 98-byte message in 99 us.
Distribution of Secure Single-hop Pair-wise Synchronization Error

- Experimental result (1 tick = 8.68 μs)

Phase II:

Secure and Resilient Global Time Synchronization

Overview

- Goal: a network-wide time synchronization.
- The algorithm.
- Local broadcast authentication
  - uTESLA (Perrig et al., IEEE S&P’01)
  - Our work: short delayed uTESLA
Secure and Resilient Global Time Synchronization Algorithm

- Each node $i$ maintains a local clock time $C_i$.
- For each neighbor node $j$, node $i$ maintains a single-hop pair-wise time difference $\delta_{i,j}$.
- A source node $S$ broadcasts its local time $C_S$ periodically.
Secure and Resilient Global Time Synchronization Algorithm

- Each node $i$ maintains a local clock time $C_i$.
- For each neighbor node $j$, node $i$ maintains a single-hop pair-wise time difference $\delta_{i,j}$.
- A source node $S$ broadcasts its local time $C_S$ periodically.
- Each direct neighbor node $i$ of node $S$ can obtain a source clock difference $\delta_{i,S}$ from node $S$ directly. Then, it broadcasts $\delta_{i,S}$.
- For other nodes, to tolerate up to $t$ malicious neighbor nodes, each node $i$ needs to obtain at least $2t+1$ source time differences through different neighbor nodes. Node $i$ chooses the median one as $\delta_{i,S}$. Then it broadcasts $\delta_{i,S}$.

Each node $i$ can estimate the global clock $C_S$ by $C_S = C_i + \delta_{i,S}$. 

---

Secure and Resilient Global Time Synchronization Algorithm

- Each node $i$ maintains a local clock time $C_i$.
- For each neighbor node $j$, node $i$ maintains a single-hop pair-wise time difference $\delta_{i,j}$.
- A source node $S$ broadcasts its local time $C_S$ periodically.
- Each direct neighbor node $i$ of node $S$ can obtain a source clock difference $\delta_{i,S}$ from node $S$ directly. Then, it broadcasts $\delta_{i,S}$.
- For other nodes, to tolerate up to $t$ malicious neighbor nodes, each node $i$ needs to obtain at least $2t+1$ source time differences through different neighbor nodes. Node $i$ chooses the median one as $\delta_{i,S}$. Then it broadcasts $\delta_{i,S}$.

Each node $i$ can estimate the global clock $C_S$ by $C_S = C_i + \delta_{i,S}$.

---

Secure and Resilient Global Time Synchronization Algorithm

- Each node $i$ maintains a local clock time $C_i$.
- For each neighbor node $j$, node $i$ maintains a single-hop pair-wise time difference $\delta_{i,j}$.
- A source node $S$ broadcasts its local time $C_S$ periodically.
- Each direct neighbor node $i$ of node $S$ can obtain a source clock difference $\delta_{i,S}$ from node $S$ directly. Then, it broadcasts $\delta_{i,S}$.
- For other nodes, to tolerate up to $t$ malicious neighbor nodes, each node $i$ needs to obtain at least $2t+1$ source time differences through different neighbor nodes. Node $i$ chooses the median one as $\delta_{i,S}$. Then it broadcasts $\delta_{i,S}$.

Each node $i$ can estimate the global clock $C_S$ by $C_S = C_i + \delta_{i,S}$.
How to Distribute Global Synchronization Messages?

- **Unicast**
  - Messages authenticated by secure peer-wise key.
  - Conclusion: too heavy communication overhead and substantial message collisions. Scalability problem

- **Broadcast**
  - Can reduce the communication overhead.
  - Require local broadcast authentication.
  - Digital signature is too expensive for sensor nodes.
  - uTESLA

Overview of uTESLA

- Perrig et al., IEEE S&P’01
- **Sender:**
  - Generate one-way key chain, \( K_i = F(K_{i+1}) \), \( 0 \leq i < n-1 \)
  - Release the commitment \( K_0 \)
  - Use key \( K_i \) for all messages sent in time interval \( I_i \), and disclose \( K_i \) in \( I_{i+1} \)

- **receiver:**
  - Security condition: the key has not been disclosed by the sender when the messages are received.
  - Verify \( K_{i} = F(K_{i+1}) \)

Using uTESLA in Time Synchronization?

- **Problems:**
  1. uTESLA itself requires loose peer-wise time synchronization.
  2. Delayed authentication causes clocks drift away again.
Short Delayed $u$TESLA

- Tight single-hop pair-wise time synchronization.
- Too short time intervals waste a lot of keys in a key chain.
- Interleaved short and long intervals.
  - Short interval ($r$): A sender broadcasts authenticated synchronization message.
  - Long interval ($R$): A sender discloses the key used in last short interval.

Short Delayed $u$TESLA (Cont.)

- Receiver:
  - Security condition: the message is sent in the sender’s last short interval.
    \[(t_i - T_0 + \delta_{\text{max}}) < i^*(r+R)+r.\]
  - Verify $K_i = F(K_{i+1})$

Security Property

- External attacks
  - Message authentication.
  - Security condition.
- Internal attacks
  - Use the median of $2i+1$ source clock differences through different neighbors to tolerate up to $i$ insiders.
**Experimental Evaluation**

- **Software package**
  - TinySeRSync
  - MICAz motes running TinyOS
  - 35 files
  - Providing 8 interfaces

- **Code size in MICAz**

<table>
<thead>
<tr>
<th>Memory</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>1961</td>
</tr>
<tr>
<td>Program</td>
<td>2414</td>
</tr>
</tbody>
</table>

- **Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># of MICAz nodes</td>
<td>60</td>
</tr>
<tr>
<td>Pair-wise sync. int.</td>
<td>4 sec</td>
</tr>
<tr>
<td>Global sync. int.</td>
<td>10 sec</td>
</tr>
<tr>
<td>Tolerance (t)</td>
<td>0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>TESLA short int.</td>
<td>10 ms</td>
</tr>
<tr>
<td>TESLA long int.</td>
<td>1 sec</td>
</tr>
<tr>
<td>Key chain length</td>
<td>100</td>
</tr>
</tbody>
</table>

**Network Deployment**

**Data Collection**

- **Sink node**
  - Broadcast reference messages to all the nodes.
  - Query each node one by one at different time.

- **All the nodes**
  - When receiving a reference message
    - Records the current global time when the message is received at MAC layer.
  - When receiving a query message
    - Send the buffered global time information to the sink node.
Synchronization Error

• Precision achieved: tens of microseconds

![Graph showing synchronization error with average and maximum error lines.](image)

1 tick = 8.68 ns

Synchronization Rate

• When t=4, 95% in 3 rounds

![Graph showing synchronization rate with percentage and tolerance lines.](image)

Communication Overhead

Number of messages per hour.

![Bar chart showing number of messages.](image)
Incremental Deployment

- Average synchronization error (left Y-axis)
- Synchronization rate when t=2 (right Y-axis)

Conclusion

- TinySeRSync:
  - Secure single-hop pair-wise time synchronization
    - Between two nodes
    - The building block of global time synchronization.
  - Secure and resilient global time synchronization
    - In the whole network
- Future work
  - Adapting the linear regression technique to compensate the constant clock drifts.