An Optimized WS-Eventing for Large-Scale Networks

Jan Skodzik, Vlado Altmann, Peter Danielis, Moritz Koal, Dirk Timmermann

University of Rostock, Germany

Institute of Applied Microelectronics and Computer Engineering
Eventing

- Easiest way to detect an event is polling
- Very ineffective
Eventing

- Avoiding polling
- Eventing (realized e.g. by publish/subscribe pattern)
Eventing for automation

- Additional requirement for automation:
  - Standardized
  - Constrained resources
  - Hard real-time behavior
  - High scalability
  - High reliability
State of the art

- We focus on Devices Profile for Web Services (DPWS)
  - Designed for devices with constrained resources
  - Intended domain (automation)

- WS-Eventing included as a standard (by OASIS)
State of the art: WS-Eventing basics

Event source

Subscription manager

Event sink

- Subscribe
- Notification
- Renew
- GetStatus
- SubscriptionEnd
- Unsubscribe
How do we find an event source?

- Usage of WS-Discovery
- Already optimized in a highly scalable manner*
- Based on the P2P network Kad(emlia)

*Vlado Altmann et al.: “A DHT-based Scalable Approach for Device and Service Discovery”
12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC14), August 2014
Standard notification procedure

- What happens if an event occurs?
- Notification transmitted sequentially
- Scales badly
- Overloading of the source

Event source

Event sinks
Highly scalable notification procedure

- Utilizing the subscribed sinks
- High scalable approach
- Forwarding the subscription list
How to achieve real-time

- For a Real-time System by choosing a platform and OS
- For communication a TDMA-based Peer-to-Peer approach has been developed called HaRTKad*
  
  - Controlled media access
  - Node has unique ID
  - Timeslot determination via ID

- We need to determine the performance of the eventing

*Jan Skodzik et al.: “HaRTKad: A hard real-time Kademlia approach”
11th IEEE Consumer Communications and Networking Conference (CCNC)
Two scenarios

- 1st scenario:
  - Sending the notification without any acknowledge (ACK)
  - Less reliability, less complexity, less data volume

- 2nd scenario:
  - A response from every notified event sinks is required
  - Higher reliability

[Diagram showing 2 scenarios with Event source and sink, notification and ACK]
EXI: Efficient XML Interchange

- In WS the data are usually XML coded
  - Not very efficient
- Usage of Efficient XML Interchange (EXI)
- EXI allows a high lossless compression
  - Binary notation of the data
- W3C standard
## Prototype

- ZedBoard as target platform
  - ARM Dual Core @ 667 MHz
  - 1-GBit Ethernet connection
- Runs the Kad software (HaRTKad application)

### Software Components

<table>
<thead>
<tr>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-Eventing</td>
</tr>
<tr>
<td>HaRTKad</td>
</tr>
<tr>
<td>lwIP (UDP/IP)</td>
</tr>
<tr>
<td>FreeRTOS</td>
</tr>
<tr>
<td>ARM-based Hardware</td>
</tr>
</tbody>
</table>
Prototype setup

● One event source and 18 event sinks
● Connected via a 1-GBit switches
Prototype setup

Triggering PC

1. Trigger to subscribe

2. Subscribe

3. Simulate Event

4. Notification

Event sink 1

4. Notification

Event sink 18
Results: Scenario 1 - Without ACK

![Graph showing linear relationship between Notification Performance $T_{\text{Dist}}$ and Number of Event Sinks. The graph illustrates an increasing trend with a label indicating a linear relationship.]
Results: Scenario 1 - Without ACK

- Linear
- Logarithmic
Results: Scenario 1 – Trend line without ACK

![Graph showing trend lines for different numbers of event sinks]

- Original
- Optimized

28 Event sinks
Results: Scenario 2 - With ACK
Results: Scenario 2 - With ACK

![Graph showing notification performance with linear and logarithmic scaling for different number of event sinks.](image-url)
Results: Scenario 2 – Trend lines with ACK
### Supported nodes at different automation scenarios

<table>
<thead>
<tr>
<th>Profile</th>
<th>Human</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle time</td>
<td>100 ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>#Event sinks per event source</td>
<td></td>
<td>200</td>
</tr>
<tr>
<td>Data amount per event source</td>
<td></td>
<td>34,240 Byte</td>
</tr>
<tr>
<td>#Event sources</td>
<td>381</td>
<td>38</td>
</tr>
<tr>
<td>#Event sinks total</td>
<td>76200</td>
<td>7600</td>
</tr>
</tbody>
</table>
How to sort the lists?

- Direct influence on who receives notification first

- 1st option:
  - First devices with high priority in timing
  - Bad results if a node close to source fails
    - As all following nodes are affected

- 2nd option:
  - First devices with highest reliability
  - Better as no following nodes are affected if a node fails
  - Better reliability and average timing
Summary

- Presentation of a scalable WS-Eventing approach
- Results from a prototype
  - An comparison with standard notification
- High efficiency due to utilization of EXI
- Two approaches sorting the lists

- Future Work:
  - Create simulation to verify behavior for thousand of nodes
Thank you for your attention!

Any questions?
Backup
## Thread prioritization

<table>
<thead>
<tr>
<th>Thread</th>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>5</td>
<td>Starts other thread before going idle</td>
</tr>
<tr>
<td>External control</td>
<td>4</td>
<td>Receiving external commands</td>
</tr>
<tr>
<td>Kad communication</td>
<td>3</td>
<td>Processing Kad packets</td>
</tr>
<tr>
<td>Search</td>
<td>3</td>
<td>Destroys search objects</td>
</tr>
<tr>
<td>Network</td>
<td>2</td>
<td>Network interface packet processing</td>
</tr>
<tr>
<td>Maintenance</td>
<td>1</td>
<td>Maintenance threads</td>
</tr>
<tr>
<td>Idle</td>
<td></td>
<td>Origin for new threads</td>
</tr>
</tbody>
</table>
**Program Flow**

<table>
<thead>
<tr>
<th>I. Initial Kad Operations</th>
<th>● Bootstrapping and Maintenance;</th>
</tr>
</thead>
<tbody>
<tr>
<td>II. Search Tolerance</td>
<td>● Determine the search tolerance</td>
</tr>
<tr>
<td>Determination</td>
<td>○ Max. one node for each hash value</td>
</tr>
<tr>
<td>III. Initial</td>
<td>● First synchronization of the Kad network</td>
</tr>
<tr>
<td>Synchronization</td>
<td></td>
</tr>
<tr>
<td>IV. Application</td>
<td>● Application on top of HaRTKad</td>
</tr>
<tr>
<td>V. Maintenance</td>
<td>● Enable maintenance of Kad network</td>
</tr>
<tr>
<td></td>
<td>○ Also Bootstrapping</td>
</tr>
<tr>
<td>VI. Re-synchronization</td>
<td>● Re-synchronize the network</td>
</tr>
<tr>
<td></td>
<td>○ Due to clock drift of nodes</td>
</tr>
</tbody>
</table>
Data Transmission in HaRTKad

![Graph showing process time versus number of transmitted integer values for read and write actions. The graph indicates an increasing process time with the number of transmitted integer values. The red line represents the read action, and the blue line represents the write action.](image-url)
Supported Network Size by HaRTKad

\[
Nodes = \frac{T_{Del}}{T_{Action} + (\log_2(Nodes) \times T_{Step})}
\]

\[
Nodes_{Max} = \left\lfloor \frac{T_{Del} \times \log(2)}{T_{Step} \times W \left( \frac{T_{Action}}{2 \times T_{Step}} \times T_{Del} \times \log(2) \right)} \right\rfloor
\]

- \(T_{Del}\): Delivery constraint
- \(T_{Action}\): Time for interacting with node
- \(T_{Step}\): Time for further search step
- \(W\): Lambert W-function
HaRTKad Prototype

- Zedboard as target platform
  - ARM Dual Core @ 667 MHz
  - 1 Gbit/s Ethernet connection
- Allows to determine $T_{\text{Step}}$ and $T_{\text{Action}}$
- Runs the Kad software (HaRTKad application)

![Diagram of software stack and communication layers]

Applications

HaRTKad

IwIP

FreeRTOS

Hardware Specification

UDP

IP

Ethernet

22.09.2014 University of Rostock