eCAL Registration / Monitoring Layer

A deep dive

http://ecal.io
About us

Kristof Hannemann

Florian Reimold
Publish / Subscribe

- Data is transferred through the network
- How do nodes know where to send the data to?
- New technique in eCAL 5.11
Idea 1: Manually

› Manual direct connections, e.g.:
  › IP + Port
  › SHM Filename

› Configuration Nightmare
Idea 2: (Central) Broker

- Broker can manage connections
Our Solution: Brokerless Zeroconf via UDP

› Use UDP Multicast to connect Nodes automatically to each other
Our Solution: Brokerless Zeroconf via UDP

- Data layer is initialized based on registration layer
Consolidation back to 1 PC

› Decreasing number of PCs, Increasing number of Nodes

› => UDP becomes a bottleneck
Introducing shared memory-based registration in eCAL
Requirements of local broadcast concept

Simple and flexible design of a local brokerless broadcast concept

Designed for high-data bandwidths on various operating systems

Automatic recovery in case of an error (no SPOF)
Idea: Broadcasting based on shared memory

1 ... N node

shm file of node X

publisher info

subscriber info

process info

common shm file

node id
timestamp

Relocatable circular queue

Registration payload
Workflow: Sending broadcasts to shared memory

- shm file of node A
  - publisher info
  - subscriber info
  - process info

- common shm file
  - node id | timestamp
  - A | 42

- shm file of node B
  - publisher info
  - subscriber info
  - process info

- node A
- node B
Workflow: Sending broadcasts to shared memory

-shm file of node A
  - publisher info
  - subscriber info
  - process info

common shm file

<table>
<thead>
<tr>
<th>node id</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>42</td>
</tr>
<tr>
<td>B</td>
<td>43</td>
</tr>
</tbody>
</table>

shm file of node B
  - publisher info
  - subscriber info
  - process info

node A

node B
Workflow: Receiving broadcasts from shared memory

shm file of node A

publisher info

subscriber info

process info

shm file of node B

publisher info

subscriber info

process info

node A

node B

common shm file

<table>
<thead>
<tr>
<th>node id</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>42</td>
</tr>
<tr>
<td>B</td>
<td>43</td>
</tr>
</tbody>
</table>
Requirements of local broadcast concept

Simple and flexible design of a local brokerless broadcast concept ✔️

Designed for high-data bandwidths on various operating systems ✔️

Automatic recovery in case of an error (no SPOF) ?
Potential SPOF of local broadcast concept

- Publisher info
- Subscriber info
- Process info

**shm file of node A**

**common shm file**

<table>
<thead>
<tr>
<th>node id</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>42</td>
</tr>
<tr>
<td>B</td>
<td>43</td>
</tr>
<tr>
<td>A</td>
<td>44</td>
</tr>
</tbody>
</table>

**shm file of node B**

- Publisher info
- Subscriber info
- Process info

**Segfault! Bye bye...**
### Automatic failure recovery

#### common shm file

<table>
<thead>
<tr>
<th>node id</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>42</td>
</tr>
<tr>
<td>B</td>
<td>43</td>
</tr>
<tr>
<td>A</td>
<td>44</td>
</tr>
</tbody>
</table>

#### shm file of node B

- **publisher info**
- **subscriber info**
- **process info**

#### node B
Requirements of local broadcast concept

Simple and flexible design of a local brokerless broadcast concept ✔

Designed for high-data bandwidths on various operating systems ✔

Automatic recovery in case of an error (no SPOF) ✔
State-of-the-art middleware

“eCAL is first pub-sub middleware that can run brokerless, zeroconf and without network stack”
How to enable the shared memory-based registration layer?
Enabling shared memory-based registration

→ Fetch 5.11.x release from https://github.com/eclipse-ecal/ecal or PPA

→ Navigate to the [experimental] section of ecal.ini config file

→ Set the option shm_monitoring_enabled to True
Thank you for your attention!
Root cause analysis: UDP bottleneck

Use-case:
Single host setups with large registration payloads

Weak spot:
UDP package fragmentation due to network MTU

Registration payload

UDP fragments

1.5kB  1.5kB  1.5kB  1.5kB  1.5kB
Root cause analysis: UDP bottleneck

Issue:
Cyclic registration update leads to recurrent UDP package bursts

→ Package bursts generates excessive CPU utilization
→ CPU peaks can cause package drops
→ **Missing UDP fragments prevent reassembling** of registration payload