Avoiding mobility-related message flooding in content-based publish/subscribe

Z. Salvador       A. Lafuente       M. Larrea

University of the Basque Country, UPV/EHU
Donostia-San Sebastián, Spain

1st International Workshop on Algorithms and Models for Distributed Event Processing (AlMoDEP 2011)
Outline

Introduction

Simple routing

Mobility support

Implementation

Conclusions
Motivation

- Traditional publish/subscribe
  - Large-scale
  - Static topology
  - Highly optimized
- Emerging applications
  - Wireless Sensor Networks
  - Mobile information dissemination
- Our scenario
  - Dynamic clients
  - Static infrastructure
  - Long-lasting subscriptions
- Our approach
  - Mobility as a primary functional requirement
  - Scalability vs. expressiveness vs. mobility support
Components

**Publishers** produce sequences of unique events and disseminate these events in the network.

**Subscribers** subscribe to receive certain kinds of events and consume events which are delivered to them.

**Brokers** mediate between publishers and subscribers (each client is connected to a single broker which is known as its *front-end broker*).

![Diagram of components](image-url)
Network

- Assumptions
  - Subscribers have unique identifiers
  - There are no cycles in the topology graph
  - Communication links are static, reliable and FIFO

- Primitives
  - `send()`: puts a message in a given communication link
  - `receive()`: waits for a message from the communication link

Messages

\[(\text{message-type}, \text{source}, \text{destination}, \text{payload}, \text{time-stamp})\]
Properties

Liveness

Every published event $e$ which matches an active subscription filter $f$ issued by a subscriber $s$ will eventually be delivered to $s$ if $s$ has not unsubscribed from $f$.

Safety

An event $e$ will only be delivered to a subscriber $s$ if (i) $e$ has been published by a publisher, (ii) $e$ matches an active subscription issued by $s$, and (iii) $e$ has not been delivered to $s$ yet.
Algorithm

\textbf{Algorithm}

\begin{algorithmic}
\When{receive(SUB, $-$, $-$, $f$, $-$) from some $z \in I_i$}
\State add $(f, z)$ to $R_i$
\Foreach{$b_j \in N_i$ such that $b_j \neq z$}
\State send(SUB, $-$, $-$, $f$, $-$) to $b_j$
\EndForeach
\EndWhen
\end{algorithmic}
Algorithm

when receive(SUB, −, −, f, −) from some $z \in I_i$ do
    add $(f, z)$ to $R_i$
    foreach $b_j \in N_i$ such that $b_j \neq z$ do
        send(SUB, −, −, f, −) to $b_j$

when receive(UNS, −, −, f, −) from some $z \in I_i$ do
    remove $(f, z)$ from $R_i$
    foreach $b_j \in N_i$ such that $b_j \neq z$ do
        send(UNS, −, −, f, −) to $b_j$
Algorithm

**when** receive(SUB, --, --, f, --) **from** some z ∈ Ii **do**

add (f, z) to Ri

**foreach** bj ∈ Ni **such that** bj ≠ z **do**

send(SUB, --, --, f, --) **to** bj

**when** receive(UNS, --, --, f, --) **from** some z ∈ Ii **do**

remove (f, z) from Ri

**foreach** bj ∈ Ni **such that** bj ≠ z **do**

send(UNS, --, --, f, --) **to** bj

**when** receive(PUB, --, --, e, --) **from** some z ∈ Ii **do**

**foreach** (f, y) ∈ Ri **such that** y ≠ z ∧
(PUB, --, --, e, --) has not been sent through y **do**

if f(e) = TRUE **then**

send(PUB, --, --, e, --) **to** y
Client mobility

Migration when a client disconnects from its front-end broker and connects to a different broker after a finite amount of time

Objective correct all of the outdated routing tables and deliver any pending events the migrating subscriber has missed

Mobility Safety
Upon the migration of any of its clients (be it publishers or subscribers), the system preserves the liveness property.

Note publisher mobility is inherently supported by the model...
Note we only preserve safety to a certain extent (2/3)...
Note broker mobility has not been addressed...
Modifications

- The use of dispatching time-stamps by front-end brokers
- A new migration request message-type: MIG
- A new event replay message-type: REP
- The use of extended routing tables
  - (Filter, Interface): local information for matching and routing
  - (Filter, Subscriber): global information to update routing tables
  - (Interface, Subscriber): global information for targeted routing
## Extended routing tables

![Routing Diagram]

<table>
<thead>
<tr>
<th>Filter</th>
<th>Interface</th>
<th>Subscriber</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>$b_1$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$b_1$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$b_1$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$b_3$</td>
<td>$s_4$</td>
</tr>
<tr>
<td>$f_5$</td>
<td>$b_3$</td>
<td>$s_4$</td>
</tr>
<tr>
<td>$f_6$</td>
<td>$b_3$</td>
<td>$s_5$</td>
</tr>
</tbody>
</table>
Algorithm

\[\text{when } \text{receive}(\text{SUB}, s, b_i, f, -) \text{ from some } z \in l_i \text{ do}\]

\[\text{add } (f, z, s) \text{ to } R_i\]

\[\text{if } s \notin C_i \text{ then}\]

\[C_i \leftarrow C_i \cup \{s\}\]

\[\text{foreach } b_j \in N_i \text{ such that } b_j \neq z \text{ do}\]

\[\text{send}(\text{SUB}, s, -, f, -) \text{ to } b_j\]
Algorithm

when receive(SUB, s, b_i, f, −) from some z ∈ l_i do
   add (f, z, s) to R_i
   if s ∉ C_i then
      C_i ← C_i ∪ {s}
   foreach b_j ∈ N_i such that b_j ≠ z do
      send(SUB, s, −, f, −) to b_j

when receive(UNS, s, b_i, f, −) from some z ∈ l_i do
   remove (f, z, s) from R_i
   if ∅(−, −, s) ∈ R_i then
      C_i ← C_i \ {s}
   foreach b_j ∈ N_i such that b_j ≠ z do
      send(UNS, s, −, f, −) to b_j
Algorithm

when receive(PUB, −, −, e, −) from some z ∈ I do
  foreach (f, y, −) ∈ R_i such that y ≠ z ∧
  (PUB, −, −, e, −) has not been sent through y do
    if f(e) = TRUE then
      if y ∈ C_i then
        ts ← clock_i()
        enqueue (PUB, −, −, e, ts) in Q_i(y)
        send(PUB, −, −, e, ts) to y
      else
        send(PUB, −, −, e, −) to y
    else
      send(PUB, −, −, e, −) to y
Algorithm

when receive(MIG, s, -, ts_last, -) from some z ∈ I_i do

if s ∉ C_i then

if z = s then

C_i ← C_i ∪ {s}

b_j ← b ∈ N_i such that (-, b, s) ∈ R_i

foreach (-, -, s) ∈ R_i do

replace in R_i (-, -, s) by (-, z, s)

send(MIG, s, -, ts_last, -) to b_j

else

C_i ← C_i \ {s}

foreach (-, -, s) ∈ R_i do

replace in R_i (-, -, s) by (-, z, s)

while Q_i(s) is not empty do

dequeue (-, -, -, -, ts) from Q_i(s)

if ts > ts_last then

send(REP, -, s, -, -) to z
Algorithm

**when** `receive(REP, −, s, −, −)` **from** some `z ∈ l_i` **do**

**if** `s ∈ C_i` **then**

```
  ts ← clock_i()
  enqueue (REP, −, −, −, ts) in Q_i(s)
  send(REP, −, −, −, ts) to s
```

**else**

```
  y ← w ∈ l_i such that (−, w, s) ∈ R_i
  send(REP, −, s, −, −) to y
```
Locality
Locality
Locality
Locality
Locality

Graph representation of nodes and connections.
Locality
Localilty

Z. Salvador

AlMoDEP 2011

15 / 21
Locality

A diagram illustrates a network topology with nodes labeled from $b_1$ to $b_{13}$, connected by lines indicating communication paths. The diagram shows the concept of locality in a network context.
Overview
# Overview

## Local Brokers
- broker_1@127.0.0.1:10000
- broker_2@127.0.0.1:20000
- broker_3@127.0.0.1:10000
- broker_4@127.0.0.1:40000

## Local Subscribers
- broker_1@127.0.0.1:10000

## Subscribers
- migrator@127.0.0.1:63182
  - broker_1@127.0.0.1:40000
  - broker_2@127.0.0.1:40000

## Filter
- b8dc7062af964b99a2807bb68b578
- d45e5863535f1795756ebc35ec2855
- 2d5b420485c1d0a1e0e06c0f8eab53

## RESET DATA
- null STR content EQU tock INT counter CQ 0

## SAVE DATA
Topology
Topology
Topology

- p
- b1
- b2
- b3
- b4
- s
Topology

The diagram represents a network topology consisting of a single source node labeled \( p \) and four destination nodes labeled \( b_1, b_2, b_3, b_4 \). The network connections are as follows:

- \( p \) is directly connected to \( b_1 \).
- \( b_1 \) is connected to \( b_2, b_3, b_4 \).
- \( b_2 \) is directly connected to \( b_3, b_4 \).
- \( b_3 \) is connected to \( b_4 \).
Execution

Graphs showing data for different brokers: broker_1@127.0.0.1:10000, broker_2@127.0.0.1:20000, broker_3@127.0.0.1:30000, and broker_4@127.0.0.1:40000. Each graph displays metrics such as Total (events/second), Incoming (events/second), and Outgoing (events/second).
Execution
Execution
Execution
Execution

Graphs showing execution data for different brokers. The graphs display metrics such as total events per second, incoming events per second, and outgoing events per second. Each graph has options to reset data and save data.
Execution
Execution

[Graphs showing execution metrics for different brokers, with details on traffic, control, routing, matching, processor, and memory.]
Execution
Execution
Execution

![Graphs showing event rates and traffic measurements for different brokers. Each graph displays the total (events/second), incoming (events/second), and outgoing (events/second) for various categories such as General, Traffic, Control, Routing, and Memory.]
Execution
Summary

- **Subscription** messages
  - are issued by subscribers
  - populate routing tables
  - flood the network

- **Publication** messages
  - are issued by publishers
  - require content-based matching
  - do not flood the network

- **Migration** messages
  - are issued by subscribers upon relocation
  - correct intermediate routing tables
  - do not flood the network

- **Replay** messages
  - are issued by brokers upon migration
  - do not require content-based matching
  - do not flood the network
Trade-offs

- **Our routing algorithm**
  - supports publisher and subscriber mobility
  - supports voluntary and involuntary mobility
  - leverages global routing information
  - does not require flooding the network
  - can exploit filter covering for event matching
  - could be used in load-balancing scenarios
  - supports prefix-forwarding optimizations

- **Our routing algorithm**
  - enforces sub-optimal propagation of subscriptions
  - sacrifices filter coverage and merging optimizations
  - requires additional data structures for global routing
Closing thoughts

• This contribution is work in progress...

• We have a reference implementation...

• It supports Android mobile devices...

• We are assessing overall scalability...