Service Discovery using OLSR and Bloom Filters

Joakim Flathagen
joakim.flathagen@ffi.no

4th OLSR Interop / Workshop, Ottawa, CA, Oct 14 2008
Presentation outline

- Who is involved
- Motivation
- Service Discovery
  - General solutions
  - MANET solutions
- Our OLSR based Service Discovery Design
  - How does it work?
  - Simulation results
  - Implementation and use
- Summary + Future work
Who is involved?

- Norwegian Defence Research Establishment (FFI) – Soldier Modernization Program (NORMANS)
  - Developing wearable computing for the future soldier
  - Command Control, Radio technology, HMI etc.
- UniK – University Graduate Center
  - Research and education institute owned by the University of Oslo (UiO) and Norwegian University of Science and Technology (NTNU)
  - 60 Phd students and 50 M.Sc students
- Researching Mobile Ad-hoc Networks aimed both for civilian and military purposes.
History – OLSR for soldier systems

- Wearable computing for soldiers 2001
- Prototypes of UHF 802.11b radios developed 2002
- UniK OLSR developed 2004 ([www.olsr.org](http://www.olsr.org))
- Field trials conducted 2005 running OLSR on UHF Radios
- New soldier systems developed and fielded 2008 (NORMANS)
Motivation

• Main goal:
  – Increasing situational awareness and reduce fratricide through automated technology
  – Reduce *user interaction* to a minimum
• Service oriented networks down to the soldier
• Every vehicle, soldier, headquarter is equipped with *services* such as:
  – Sensors: Chemical detectors, battery indicator, HR-monitor
  – Tools: Laser Range Finder, Night vision, Map-server
  – People: Squad leader, commander.
• Our aim is bandwidth constrained MANETs (< 100Kbps)
• Also applicable to first responder networks
Service Discovery

• Aim: Find the IP-addresses of services and resources without user interaction.
• General service discovery solutions are not applicable to MANETs:
  – Bad scalability
  – To much overhead
  – Relies on directories
• Tailor made solutions must be provided for MANETs
Service Discovery in MANETs

- Proposed solutions differ in many aspects:
  - Service Descriptors
    - WSDL, GUID / Integer, Text, Bloom Filters
  - Architecture
    - Directory, Directory-less, Hybrid
  - Discovery mode
    - Proactive, Reactive
  - Layer
    - Application Layer Service Discovery
      - (SLPManet, PDP, Konark, Sailhan)
    - Cross-Layer service discovery
      - AODV (SEDRIAN, Engelstad)
      - OLSR (Li, Jodra)
Our proposal - Mercury

- **Directory-Less**
  - Fully Distributed
  - Reactive discovery *with a proactive taste*
- Service dissemination using **OLSR**
- Service descriptors defined as **Bloom Filters**
- **Caching** to reduce overhead and discovery latency
Mercury - Functionality

- Mercury provides an interface to provide service discovery for all applications on the computer.
- Each node employs three repositories:
  - Advertised services (distributed on-demand)
  - Foreign services (caching incoming advertisements)
  - Requested services
How it works

- Services are registered locally by all applications
- Service are advertised on request
- Service requests are disseminated if the cache is empty for the given service (or if the app searches for ALL services).
- All service advertisements are cached for a period of time.
Mercury Protocol

- OLSR default forwarding algorithm ensures backwards compatibility – Not every node needs to run Mercury.
Bloom Filters

- Bloom Filters allows data-representation in a space efficient manner by hashing service descriptors.
- As an effect, Bloom Filters allow *false positives*
Bloom Filters

- Bloom Filters are used to represent all services.
- Distribute a series of services: Space efficient, predictable, and does not interfere with OLSR operation.

```
<table>
<thead>
<tr>
<th>Advertised services</th>
</tr>
</thead>
<tbody>
<tr>
<td>”Gateway” = {1,4,5}</td>
</tr>
<tr>
<td>”Printer” = {2,4,6}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>0 _ 1 _ 0 _ 1 _ 1 _ 0 _ 1 _ 1</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>1 _ 1 _ 0 _ 0 _ 0 _ 1 _ 1 _ 1</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 _ 1 _ 1 _ 0 _ 1 _ 0 _ 1 _ 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>1 _ 1 _ 0 _ 0 _ 0 _ 1 _ 1 _ 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>0 _ 0 _ 1 _ 0 _ 1 _ 1 _ 1 _ 0</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 _ 0 _ 0 _ 1 _ 1 _ 0 _ 1 _ 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>0 _ 1 _ 0 _ 1 _ 1 _ 1 _ 0 _ 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>1 _ 0 _ 0 _ 1 _ 1 _ 1 _ 0 _ 1</td>
</tr>
</tbody>
</table>

```

Mobile Ad hoc Network
Simulation Results - Overhead

- Average discovery process compared to App-level SD
Simulation Results - Delay

![Graph showing simulation results for delay with different lines for SLPManet, PDP, and Mercury, and error bars indicating variability.](image-url)
Caching

- Caching in mobile networks are challenging as the cache may be false due to mobility.
- We propose a path-aware caching approach
  - Consults the OLSR routing table on service lookup
  - Reduces false positives to application
  - However, the routing table may still be false.
Simulation Results – Caching effect

- Simulations show the probability of the cache to return false positives upon a query.
- Using a path-aware cross-layer approach, the false positive probability *almost eliminated*
OLSRd implementation

- Mercury is implemented as a plugin to OLSRd (www.olsr.org)
- Applications connect, advertise, discover and withdraw services using a simple socket interface
Discovery of SIP UAs

- Using the Inter-process communication interface, any existing distributed application can be extended to utilize Mercury.
- Simple socket Code.

```java
mySD = new Socket("localhost", port);
out = new PrintWriter(mySD.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(
    mySD.getInputStream()));
out.println("ADVR SIP");
out.println("RQST SIP ALL");
```
Summary & Future work

- OLSR facilitate cross-layer support of Service Discovery
- Bloom Filters is an flexible and efficient way to distribute services.
- Caching is beneficial (use with caution).
- The OLSRd plugin library is an efficient way to implement service discovery.

Future work:
- Real-world trials.
- Optimize OLSR and SD settings for bandwidth-constrained environments
- Examining different movement patterns for simulation
References


References

References