SpartanRPC
Secure WSN Middleware for Cooperating Domains

Peter Chapin and Christian Skalka
University of Vermont

MASS-2010; San Francisco; November 8-12, 2010
Outline

• What Problem are we Solving?
• SpartanRPC
• Language Design and Implementation
• Performance Evaluation
• Future Work
What Problem are we Solving?
Example: Emergency Responders

• Fire fighter’s sensor network
  • Senses fire conditions

• Medical sensor network
  • Tracks state of injured individuals*

• Police with hand held devices
  • Communicates with other networks to alert officers about fire conditions

SpartanRPC

• Extension of the nesC programming language

• *Link level* RPC discipline
  • Hides details of radio communication
  • Extends nesC “wiring” over the air
  • Allows dynamic reconfiguration of internode wiring

• Provides access control feature
  • Hides authorization computations
  • Allows multiple roles in the same network

• Intended to support higher level protocols
  • A light weight infrastructure
SpartanRPC Example

Interface

```java
interface LEDControl {
    duty void setLeds(uint8_t ctl);
}
```

Client

```java
module LoggerC {
    uses interface LEDControl;
}
implementation {
    ...
    post LEDControl.setLeds(42);
    ...
}
```

Server

```java
module LEDControllerC {
    provides remote interface LEDControl {
        requires "LEDMASTER";
    }
    implementation {
        ...
        duty void LEDControl.setLeds(uint8_t ctl) {
            ...
        }
        ...
    }
```
configuration AppC { }
implementation {
    components ClientC, RemoteSelectorC;
    ...
    auth "LEDMASTER"
    ClientC.LEDControl -> [RemoteSelectorC].LEDControl;
    ...
}
SpartanRPC Addressing

\((N, C, I)\)

- \(N\) : TinyOS node ID (0xFFFF for broadcast)
- \(C\) : Component ID
- \(I\) : Interface ID

\((C, I)\) must be agreed upon; “well known” service ID
Sprocket

• Rewrites SpartanRPC programs to plain nesC
• Roles associated with symmetric AES keys
  • (role, key) associations known statically
• Authorization via simple MAC
  • 32 bit CBC-MAC computed with AES encryption
• Alternate authorization algorithms possible
  • Authorization independent of SpartanRPC features
## Performance Evaluation (Memory)

<table>
<thead>
<tr>
<th></th>
<th>ROM Bytes</th>
<th>ROM %</th>
<th>RAM Bytes</th>
<th>RAM %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Client</td>
<td>13096</td>
<td>-</td>
<td>378</td>
<td>-</td>
</tr>
<tr>
<td>Baseline Server</td>
<td>12576</td>
<td>-</td>
<td>306</td>
<td>-</td>
</tr>
<tr>
<td>Duties Client</td>
<td>13568</td>
<td>3.6</td>
<td>398</td>
<td>5.3</td>
</tr>
<tr>
<td>Duties Server</td>
<td>12624</td>
<td>0.4</td>
<td>308</td>
<td>0.6</td>
</tr>
<tr>
<td>Security Client</td>
<td>22662</td>
<td>73</td>
<td>608</td>
<td>61</td>
</tr>
<tr>
<td>Security Server</td>
<td>21978</td>
<td>75</td>
<td>534</td>
<td>74</td>
</tr>
</tbody>
</table>

Tmote Sky nodes with TI’s MSP430F1611 controller
48 KiB ROM, 10 KiB RAM
### Performance Evaluation (Energy)

<table>
<thead>
<tr>
<th></th>
<th>Compute Burst (ms)</th>
<th>Compute Burst (μJoule)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Client</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>Duties Client</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>Security Client</td>
<td>4</td>
<td>22</td>
</tr>
</tbody>
</table>

Transmitter pulse lasted 15 to 16 ms => 780 μJ
MSP430F1611 draws max 500 μA => 6μJ in security case
Future Work

• Confidentiality support
• Access control using trust management concepts
• General staged programming in embedded systems

Peter Chapin (pchapin@cem.s.uvm.edu)
Christian Skalka (skalka@cem.s.uvm.edu)

Sprocket home page: http://www.cs.uvm.edu/~pchapin/Sprocket
Overflow Slides
Component Managers

```c
typedef struct {
    uint16_t node_id;   // N
    uint8_t local_id;   // C
} component_id;

typedef struct {
    int count;
    component_id *ids;
} component_set;

interface ComponentManager {
    command component_set elements();
}
```

Components used to specify dynamic wire endpoints provide ComponentManager
RPC calls are multicast
Sprocket

Transforms SpartanRPC programs into ordinary nesC programs

Stub generated for each dynamic wire. Skeleton for each remote interface.
Dynamic Wire Conversion

ClientC.LEDControl -> [RemoteSelectorC].LEDControl

The above dynamic wire is converted by Sprocket into the configuration

```
components Spkt__1;
ClientC.LEDControl -> Spkt__1;
Spkt__1.ComponentManager -> RemoteSelectorC;
Spkt__1.PACKET -> AMSenderC;
Spkt__1.AMPacket -> AMSenderC;
Spkt__1.AMControl -> ActiveMessageC;
Spkt__1.AMSend -> AMSenderC;
```

Where Spkt__1 is the stub that prepares the data packet and marshals arguments. Duty post operations in ClientC are converted into command invocations.
Interface ID (4 bits)
Duty ID (4 bits)
Remote node ID (8 bits)
N -1 additional ID pairs
32 bit MAC (optional)
Number of Components (8 bits)
Remote component ID (8 bits)
Arguments

Overhead = 2N + 2 bytes (or 2N + 6 when MAC is used)
Directed Diffusion Example

```c
interface InterestManagement {
    duty void set_interest(
        uint16_t sender_node,
        int temp_threshold,
        int interval,
        int duration);
}
```

```c
interface DataManagement {
    duty void set_data(
        int sender_node,
        int originator_node,
        int temp_value);
}
```