Towards Next Generation Wireless Sensor Networks

Rethinking Middleware Design

Paolo Costa
costa@cs.vu.nl
http://www.cs.vu.nl/~costa
Vrije Universiteit Amsterdam, NL

MidSens, 28th November 2007
Outline

1. Next Generation WSN
2. Middleware State-of-the-art
3. Middleware Reloaded
4. Case Example: TeenyLIME
5. Your Turn
6. Conclusions
Current Trend

- Wildlife monitoring
- Habitat monitoring
- Healthcare
- Smart ambient

Wireless Sensor and Actuator Networks

Centralized vs. Decentralized
Homogeneous vs. Heterogeneous

28/11/2007
New Vision

- The system scale is rapidly increasing
  - Ultra Large Scale
  - “systems of systems”
  - millions (if not billions) of nodes
- Pervasive computing is becoming reality
  - devices everywhere
  - heterogeneous networks (including wired)
- Wired and wireless integration
Challenges

- **Scalability**
  - add /remove resources without breaking the systems

- **Decentralization**
  - the control logic must be embedded in the network
  - coordination among multiple tasks

- **Resource scarcity**
  - energy, CPU speed, memory...

- **Heterogeneity**
  - different hardware
  - different scenarios

- **Dynamicity**
  - devices fail, new devices join
  - new functionality to address dynamically arising requirements
MW for WSNs? Of course

- Increasing complexity naturally call for middleware
- Alleviate the burden on programmers
  - powerful abstractions
  - efficient implementations
- Sounds easy but...
MW for WSNs? Not Really

- Unfortunately, the research on middleware for WSN is still limited
- Middleware conference
  - in eight editions, only 3 papers [2003, 2004, 2007]
- Survey [K.U. Leuven]
  - only 14 papers (out of 126 references) refer to “middleware” in the title
- Google “middleware sensor networks”
  - mainly position papers (issues, challenges, ...)
- MidSens experience
  - difficulty in involving the community
How Come?

- In part, it is due to “social” reason
- WSNs start within the network community
  - Need to coordinate efforts from software engineering to networking, going through distributed systems
- However, there are also technical issues…
Bell’s Law and IT

“Roughly every decade a new, lower priced computer class forms based on a new programming platform, network, and interface resulting in new usage.”

--- Gordon Bell

(adapted from D. Culler)
Historical Background

- **50s**
  - application running on a single PC
  - no OS layer

- **60s**
  - application running on a single PC
  - OS abstracts from hardware

- **70s**
  - Networked applications
  - Network stack appears
Historical Background

- **80s**
  - Distributed applications
  - Middleware abstracts from network

- **90s**
  - Loosely coupled applications
  - Middleware services

- **2000s**
  - Embedded applications
  - ???
State-of-the-art

- **Middleware:**
  - Database
  - Mobile Agents
  - Pub/Sub
  - Shared Memory
  - Micro vs. Macro Programming

- I also contributed to increase the noise...
  - Publish/Subscribe [MASS’05]
  - Runes (Component-based) [PERCOM’07]
  - TeenyLIME [MidSens’06, Middleware’07]
State-of-the-art

- Despite the effort, programming and engineering WSNs is still cumbersome
- Assembling different solutions is hard (if not impossible)
  - e.g., MAC / routing / abstraction
- Most real applications are built from scratch
  - lack of reusable software
- Too many possible combinations
  - hw / sw / application domain / environment
- Key Issue
  - combining abstraction with efficiency
Design Flaws

- **Layered Approach**
  - non-functional requirements (e.g., energy-efficiency) need cross-layer interactions
  - also among research communities ;)

- **Distribution Hiding**
  - coarse-granularity prevents efficient usage of resource

- **Application-agnostic Design**
  - hard to address heterogeneity

- **High Level Programming Abstractions**
  - inflexible implementation
Requirements

- Middleware is key to tackle the challenges of next-generation WSNs

- Potentially contradictory goals:
  - generality / re-usability
  - efficiency

- MW should provide
  - high-level abstraction
  - high flexibility
  - fine-grained granularity

- **Approach:**
  - don’t sweep complexity under the carpet
  - but deal with it
Let’s Get Started

IT WOULD TAKE US TWO YEARS TO BUILD A PRODUCT THAT WILL BE OBSOLETE ONE YEAR FROM NOW.

OKAY, LET’S GET STARTED.
From One to Many

- Middleware must extend its scope also to networking components
- Complex middleware assembled from simple middleware components
- Complexity should emerge from the interaction of simple components
- **Micro-kernel**-like approach
  - monolithic approaches hardly scale on OS, let apart large-scale networks
  - decompose complex issues in simpler ones
Distribution Awareness

- Experience in wired settings teaches us that distribution cannot be fully hidden
  - shared memory vs. message passing
  - remote invocation
    - latency issue
    - RMI message passing
- Goal: make distribution easy
  - abstract from low-level sockets
  - but still give the control
Co-design

- Currently components (protocols) are developed in isolation
- Blind optimizations may become harmful for the whole system
  - local efficiency vs. global efficiency
- Joint design of the middleware chain
- Address separate requirements at the same time
Co-design: MAC vs. Routing

- **Currently:**
  - MAC protocols aim at optimizing one-hop communication and resource usage
  - Routing protocols aim at efficiently disseminating messages, (mostly) regardless the underlying MAC

- **Future (?):**
  - MAC should be designed also on the basis of what the routing needs
  - e.g., gossip-based protocols and tree-based protocols have different requirements on MAC
Application Awareness

- Each component should be designed against a given application domain

- Role-based Design
  - different application roles should be reflected in different component roles

- Example: publish/subscribe
  - all nodes are potentially publisher/subscribers
    - not true in reality
  - if the role is known, routing can be made efficient
Application Hooks

- Components should be designed such that applications can easily interact with
- Applications should not tailor the component themselves
  - otherwise middleware would be pointless
- Applications specify requirements but it is up to the component to meet them
  - tailor the behavior according to the application’s recommendations
Example: TDMA MAC Design

- TDMA MAC protocols split the time frame in multiple slots assigned to different nodes.
- Ideally every node should have a chance to talk to and listen to others.
- In reality, traffic patterns strongly depend on application profile:
  - Actuators are mainly receivers.
  - Sensors are mainly senders.
- MAC protocol should reconfigure itself according to the application role.
Macro-programming

- The programmer to a large extent does not “see” the single nodes, but only their aggregate effects in the network
  - useful to diminish the burden on programmers
  - involve domain experts
- Flexible implementation required
  - must adapt to heterogeneous context
  - plug-in functionality
  - abstractions should support application-driven design both at language and system level
Feasible ?

SALES ARE DROPPING LIKE A ROCK.

OUR PLAN IS TO INVENT SOME SORT OF DOOHICKEY THAT EVERYONE WANTS TO BUY.

THE VISIONARY LEADERSHIP WORK IS DONE. HOW LONG WILL YOUR PART TAKE?
TeenyLime in a Nutshell

- In traditional networks support for software development comes in the form of middleware.
- Complex coordination patterns must be expressed involving both proactive and reactive interactions.
- Based on the current state of the system.
- Need for data-centric, state-based programming abstractions.
- To support this scenario, we devised a novel tuple space model, called TeenyLime.
  - Separating application logic and communication.
  - Tuple space operations are used for:
    - Data collection.
    - Device coordination.
TeenyLIME Model

- Each node hosts a tuple space
- Core abstraction: transiently shared tuple space
  - the tuple space “changes” based on connectivity
  - shared tuple space view is different at each node
- Components...
  - write to their own space or neighbor’s ones
    - sensor data or application data
  - read from their own space or neighbor’s ones
  - react to tuples from their own or neighbor’s ones
Micro-kernel

- TeenyLIME aims at providing a minimal set of APIs to support sense-and-react applications
- Many features are not included on purpose
  - keep it simple
  - they can be added on top
Micro-kernel

```c
interface TupleSpace {
    command TLOpId_t out(bool reliable,
                          TLTarget_t target, tuple *tuple);
    command TLOpId_t rd(bool reliable,
                         TLTarget_t target, tuple *pattern);
    command TLOpId_t in(bool reliable,
                         TLTarget_t target, tuple *pattern);
    command TLOpId_t rdg(bool reliable,
                          TLTarget_t target, tuple *pattern);
    command TLOpId_t ing(bool reliable,
                         TLTarget_t target, tuple *pattern);
    command TLOpId_t addReaction(bool reliable,
                                 TLTarget_t target, tuple *pattern);
    command TLOpId_t removeReaction(TLOpId_t operationID);

    event result_t tupleReady(TLOpId_t operationId,
                               tuple *tuples, uint8_t number);
    event result_t reifyCapabilityTuple(tuple *capTuple,
                                         tuple* pattern);
}

interface SystemTupleSpace {
    event tuple* reifyNodeTuple();
}
```
Micro-kernel

- TeenyLIME aims at providing a minimal set of APIs to support sense-and-react applications
- Many features are not included on purpose:
  - keep it simple
  - they can be added on top
- Currently available:
  - Multi-hop geographic routing (GPRS-like)
  - Object Tracking
  - Mutual Exclusion
Distribution Awareness

- Despite the tuple-space abstraction, developers are always in control of distribution.
- Scope restrictions specified as:
  - TL_NEIGHBORHOOD - any one hop neighbor
  - TL_LOCAL - in the tuple space of this node
  - TL_ANY - neighborhood+local
- Distribution is not hidden but abstracted from low level network details.
Co-design

- Ongoing research focuses on a tight interaction with MAC, striving for efficiency
- TeenyLIME traffic patterns exhibit peculiarity
  - periodic reaction refresh
  - broadcast rd / in operation
  - unicast replies
- Currently based on 802.15 MAC
- Future (?): tailored protocol
Application Awareness

- Domain specific middleware
  - sense-and-react applications

- Originally TeenyLIME used to have a statically defined tuple structure
  - computational overhead (matching)
  - memory overhead (storage)
  - network overhead (message)

- Solution: auto-generated structure / procedure
  - applications specify data format actually used
  - code generator creates proper code
Things We Left Out

- Beside middleware design, there are a number of functionality worth investigation e.g.,
  - code update
  - integration of different infrastructure
  - security
    - multiple applications running on a single node
  - ...

Conclusions (?)

- Involve people from SE and Networking in the loop
- Think global!
- One fits all is a holy grail but we do not need to re-invent the wheel
- Let’s face complexity by dealing with in smaller pieces rather than hiding it
How About You?

- Life Cycle Support for Sensor Network Applications
  - include distribution, application hooks
- On the Integration of Sensor Networks and General Purpose IT Infrastructure
  - how to deal with conflicting requirements
- A Node Discovery Service for Partially Mobile Sensor Networks
  - exploit application knowledge
- A PiggyBacking Approach to Reduce Overhead in Sensor Network Gossiping
  - Gossip-aware MAC protocol design
- Energy-Aware Compilation for Wireless Sensor Networks
  - extend the model to become application-aware as well
  - e.g., traffic pattern
Thank You