Middleware Approaches for Sensor Networks

Summer School on WSNs and Smart Objects
Schloss Dagstuhl, Aug. 29th – Sept. 3rd, 2005

Dr. Pedro José Marrón

pedro.marron@informatik.uni-stuttgart.de

University of Stuttgart

IPVS, Distributed Systems Group

Outline

- Motivation
- Challenges in the development of middleware solutions
- Classification of middleware systems
  - Classic middleware
  - Data-centric middleware
  - Virtual Machines
  - Adaptive middleware
- Comparison
- Conclusion

Sensor Network Applications

- Habitat Monitoring Applications
  - Great Duck Island (GDI) System
  - Hogthorb – Sow heat period monitoring
- Environment Observation and Forecasting Systems
  - ALERT – National Weather Service
  - Floodnet – River monitoring in UK
- Health Applications
  - Care in the Community – UK
  - UbiCare – UK
- Military Applications
  - WINS – Surveillance and exploration
  - Odyssey – Underwater surveillance

Sensor Network Applications

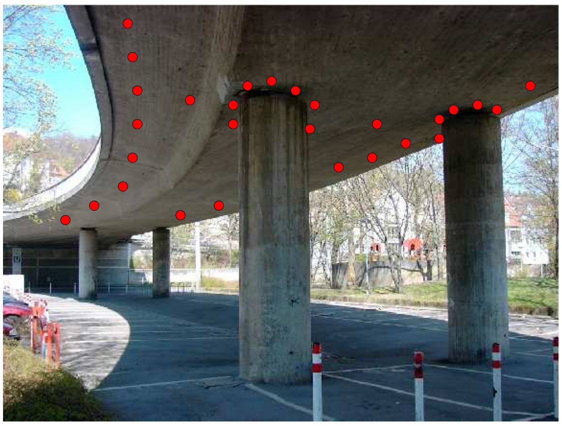
- Intelligent Building Monitoring
  - Structure Health Monitoring System – US, Canada
  - Sustainable Bridges – EU
- Intelligent Traffic Systems
  - Safe Traffic – Sweden
  - Vehicular Networks (CarTalk 2000) – EU
- Smart Room Environments
  - Aware Home – Georgia Institute of Technology
  - Sense-R-Us – University of Stuttgart
- . . . and many more

Sensor Network Applications

- Intelligent Building Monitoring
  - Structure Health Monitoring System – US, Canada
  - Sustainable Bridges – EU
- Intelligent Traffic Systems/Vehicular Networks
  - Safe Traffic – Sweden
  - Vehicular Networks (CarTalk 2000) – EU
- Smart Room Environments
  - Aware Home – Georgia Tech
  - Sense-R-Us – University of Stuttgart
- . . . and many more

Sustainable Bridges

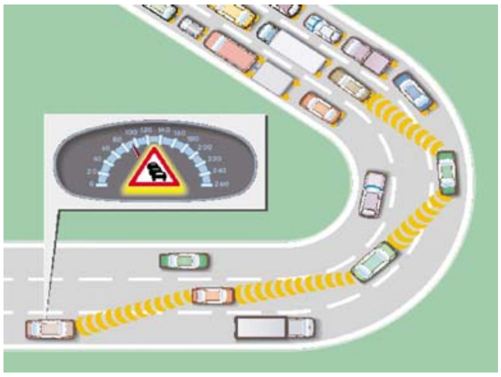
- **Goal:** Cost-effective monitoring of bridges to detect structural defects

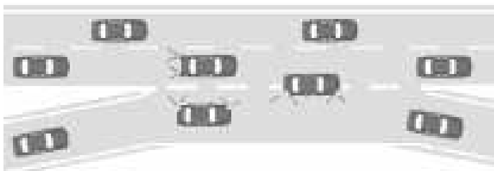


- Static sensor nodes
- Simple and complex data: temperature, vibration
- Noise detection and localization
- Data sampling: 40 KHz!
- Time synch.: 60 μs
- Sensor lifetime: 3 years!
- Hybrid network topology

Vehicular Networks – CarTalk

- **Goal:** Development of a cooperative driver assistance system
- Provide an Ad-Hoc warning system for:
  - Traffic jams
  - Accidents
  - Lane/highway merging
- Standard query interface:
  - Avg speed/temperature, road conditions
  - Location, position





Vehicular Networks – Properties

- Wide range of sensor data continuously gathered
  - Speed, position, tire pressure
- Sensor data is highly dynamic
- Sensors located within the car
- Communication plays a crucial role in the system
- Processing of data must be performed in a timely manner
- Energy constraints are not so important
- Sensor nodes are mobile
- Ad-hoc network topology

Application Commonalities

Most sensor network applications:

- Are **data-centric** and/or data-driven
  - Provide some form of monitoring
- Are **state-based**
  - Their needs might change depending on the current state of the application
- Must be **fault-tolerant** with respect to failures and/or environmental conditions
- Require **high availability** of sensors and nodes
- Must be either flexible or **reconfigurable**

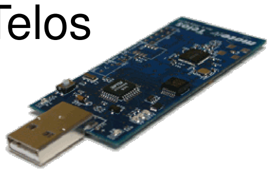
Application Differences

Property	Sust. Bridges	VANETs
Data Model	Specific	Generic
Query Model	Push-based	Pull-based
Prog. Paradigm	Pub/Sub	Query-based
Topology	hybrid	ad-hoc
Dist. Transparency	○	●
Energy	●	◔
Mobility	○	●
Real-time	●	◑
Time Synch.	◕	◔
Reconfiguration	◔	◕

○ Not important ◑ Medium ● Very important

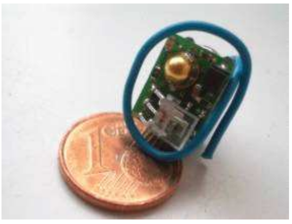
Hardware Platforms

Moteiv Telos



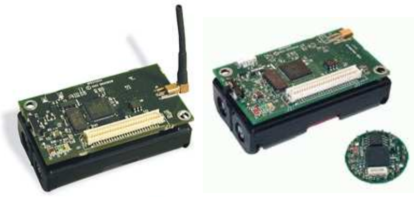
Smartdust

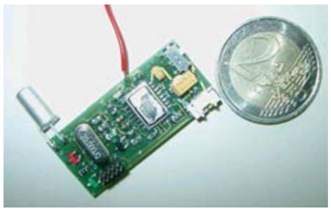
BTNode



Teco Particle

Crossbow MICAs





Teco Node

Problems to Solve

- Redundancy and reimplementation of code
- Similar abstractions for many kinds of applications
- In the presence of:
  - Highly heterogeneous applications
  - Highly heterogeneous hardware platforms
  - Very different algorithmic complexity

Middleware to the rescue!

Challenges of Middleware Systems

- **Abstraction support:**
  - Hide the complexity of each individual node and provide a holistic view of the network
  - Data-centric, publish-subscribe, event systems
  - Support a wide range of applications and hardware platforms
- **Efficiency:**
  - Be energy efficient and “resource-friendly”
  - Have cross-layer capabilities for optimization
- **Programmability:**
  - Provide support for configuration and reconfiguration
  - Policy creation and distribution

Challenges of Middleware Systems

- **Adaptability:**
  - Support for algorithms with adaptive performance characteristics (Adaptive fidelity algorithms)
  - Reactive adaptation requires system monitoring
- **Scalability:** On the number of nodes, users, etc.
- **Topology:** Optimal type of network configuration
  - ad-hoc, infrastructure, hierarchical, hybrid
- **Security:** Regarding data processing, data communication, device tampering, etc.
- **Non-functional properties (QoS):**
  - Timeliness, availability, fault-tolerance

Classification of Middleware Systems

- One possible way is to concentrate on the type of abstraction level
- **“Classic”:** Hide the complexity of network communication and data transfer
- **Data-centric:** Provide the abstraction of the network as a database
- **Virtual Machines:** The network is a collection of code interpreters that take care of running programs/scripts
- **Adaptive:** Main focus is on adaptability
- Let us look at current middleware systems

Classification of Middleware Systems

“Classic”	Data-centric	Virtual Machines	Adaptive
Impala	Cougar	Maté	MiLAN
TinyLime	TinyDB	Smart Messages	AutoSeC
EnviroTrack	DSWare	Agilla	TinyCubus
Mires	SINA	SensorWare	
Hood			

- Only the most relevant projects are listed in this table

Classification of Middleware Systems

“Classic”	Data-centric	Virtual Machines	Adaptive
Impala	Cougar	Maté	MiLAN
TinyLime	TinyDB	Smart Messages	AutoSeC
EnviroTrack	DSWare	Agilla	TinyCubus
Mires	SINA	SensorWare	
Hood			

“Classic” Middleware

Features of “classic” middleware

- Usually provide abstractions regarding:
  - Communication primitives
  - Communication paradigms (e.g. publish/subscribe)
  - Application requirements
- Some give more importance to re-programmability and adaptation
- Similar topology consideration, although mobility and scalability are still hard issues
- Most “classic” middleware projects are not concerned about security and QoS

Features in more Detail

	Impala	TinyLime	EnviroTrack
Abstraction	communication, code installation	tuplespace, data sharing	tracking
Efficiency	energy, cross-layer	energy	energy
Programmability	versioning, event-based	one-time	one-time
Adaptability	state-machine	data loss	
Scalability	herd-size, iPAQ		
Topology	ad-hoc, mobile	ad-hoc	hierarchical
Security			
QoS		fault-tolerance	fault-tolerance

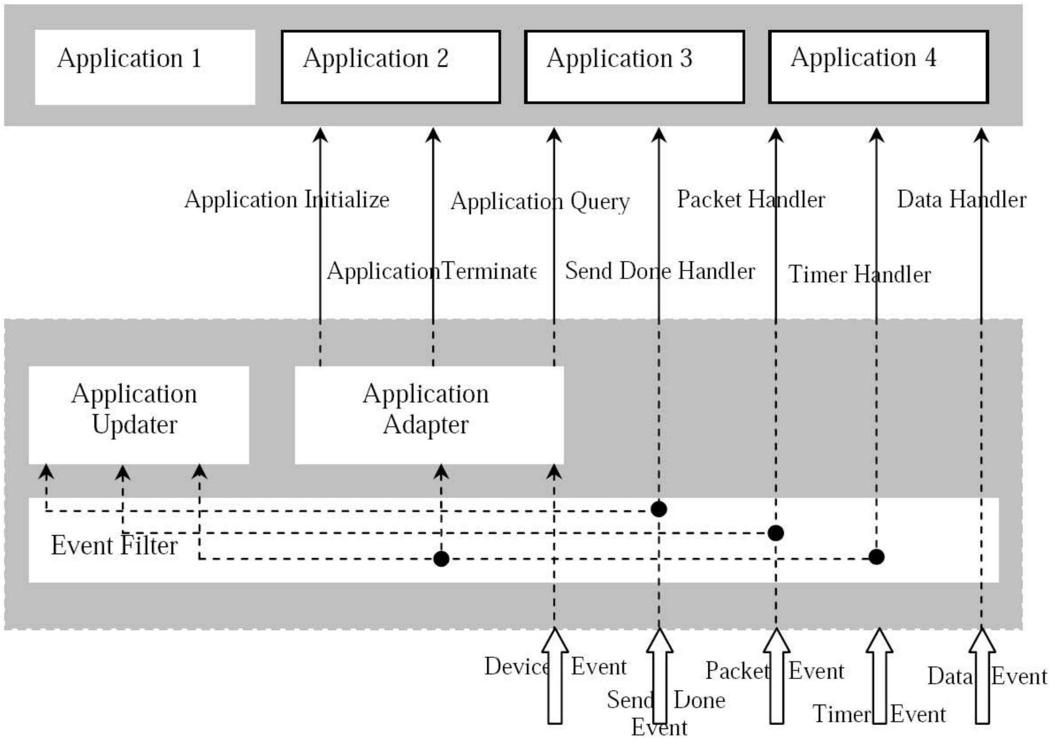
Features in more Detail

	Mires	Hood
Abstraction	pub/sub, message-oriented	neighborhood
Efficiency	energy	data caching
Programmability	one-time, topics	one-time
Adaptability		parameterization
Scalability		maximum number of neighbors
Topology	multi-hop	single-hop
Security	(planned)	
QoS		

The Impala Middleware

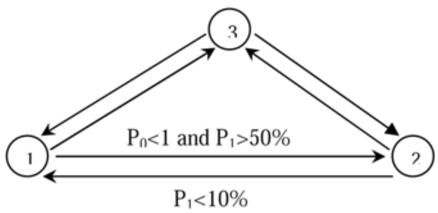
- **Goal:** Ensure reliability and ease of upgrades for long-running sensor network applications
- **Philosophy:** Mobile (wild) environments require continuous fine-tuning
- **Methodology:**
  - Event-based programming model
  - Implementation as part of the ZebraNet project
  - Design rationale:
    - Modularity
    - Correctness
    - Ease of Updates
    - Energy efficiency

Impala Architecture



Application Adapter

- Adaptation is required to:
  - Increase performance by re-parameterizing the application
  - Improve robustness choosing alternative protocols in case of hardware failures
- Adaptation Finite State Machines are used for parameter-based adaptation
  - $P_0 =$ Avg. num. neighbors
  - $P_1 =$ battery level



- Device-based adaptation is performed on the basis of Application Device Tables

Application Updater

- Must be able to handle the following issues:
  - Incomplete updates
  - On-the-fly update of code while executing
  - Contemporaneous updates
  - Inconsistent updates
  - Propagation protocol
  - Code memory management
- Approach taken by the updater:
  - Linking performed on the nodes
  - Use of version numbers
  - Epidemic software transmission

Evaluation of Impala

- Advantages
  - Robust code update mechanism that ensures the reliability of long-running applications
  - Provides adaptation capabilities
  - On-the-fly updates
  - Fault-tolerance
- Limitations
  - Heterogeneity is not an issue
  - Adaptation is limited to the capabilities of the state machine
  - Application domain is rather simplistic

Data-centric Middleware

Features

- Abstractions revolve around data and not communication
  - Database-like abstractions
  - Specially designed for sensor networks
- Focus on efficient evaluation of query plans
- Most rely on some form of SQL-like language
- Adaptation and reconfiguration is for most projects not an issue
- Injection of queries from outside the network
- Mostly no consideration of security or QoS issues

Features in more Detail

	Cougar	TinyDB	DSWare	SINA
Abstraction	database	database	real-time data service	distributed database
Efficiency	energy, multi-query plan	energy, query plan	energy	
Programmability	SQL	SQL, aggregation	SQL, events	SQTL
Adaptability				
Scalability	multiple queries			location-aware
Topology	base station	base station		
Security				
QoS			real-time, reliable storage	

The TinyDB Middleware

- **Goal:** Development of an acquisitional query processor layer for sensor networks
- **Philosophy:**
  - "Efficient data acquisition is our business"
  - "Only continuous queries are important"
- **Methodology:**
  - Implementation as a component of TinyOS
  - Definition of an acquisitional query language (ACQL)
  - In-network query processing and classification of query types
    - Reduce communication overhead
    - Reduce energy consumption

Acquisitional Query Language

- **Data model:**
  - Entire sensor network is a single table
  - Columns contain all the attributes in the network
  - Rows specify the individual sensor data
- **Query model:**
  - All queries create a continuous data stream
  - Query language is SQL-based with new language features
    - Traditional SQL with aggregation operators
    - Event processing capabilities
    - Creation of storage points
    - Specification of lifetime queries

ACQL Examples

- Event-based queries:

```
ON EVENT bird-detector(loc)
    SELECT AVG(light), AVG(temp), event.loc
    FROM sensors AS s WHERE dist(s.loc, event.loc) < 10m
    SAMPLE INTERVAL 2s FOR 30s
```

- Storage-based queries:

```
CREATE
    STORAGE POINT recentlight SIZE 5s
    AS (SELECT nodeid, light FROM sensors SAMPLE INTERVAL 2s)
```

- Lifetime-based queries:

```
SELECT nodeid, accel
    FROM sensors
    LIFETIME 30 days
```

Query Processing

- TinyDB performs power-based optimizations
  - Metadata sent periodically to the sink for optimization
  - Ordering of sampling and predicates
  - Event query batching
- For processing, TinyDB uses Semantic Routing Trees (SRTs)
  - Choice of parent based on semantic information
  - Index implemented as a network overlay
  - Flooding to announce query
  - Parent selection

Query Processing (cont.)

- Performed in two steps:
  - Sampling and local operator execution
  - Data propagation
- Sampling step
  - Allow nodes to sleep for as much of each epoch as possible
  - Computation of a partial state record
- Data propagation
  - Prioritized based on three schemes: *naive*, *winavg* and *delta*
  - Adaptation of transmission and sampling rate

In-network Aggregation Framework

- TinyDB supports aggregation functions conforming to:

$$\begin{aligned} Agg_n &= \{f_{init}, f_{merge}, f_{evaluate}\} \\ f_{init}\{a_0\} &\rightarrow < a_0 > \\ f_{merge}\{< a_1 >, < a_2 >\} &\rightarrow < a_{12} > \\ f_{evaluate}\{< a_1 >\} &\rightarrow \text{aggregate} \end{aligned}$$

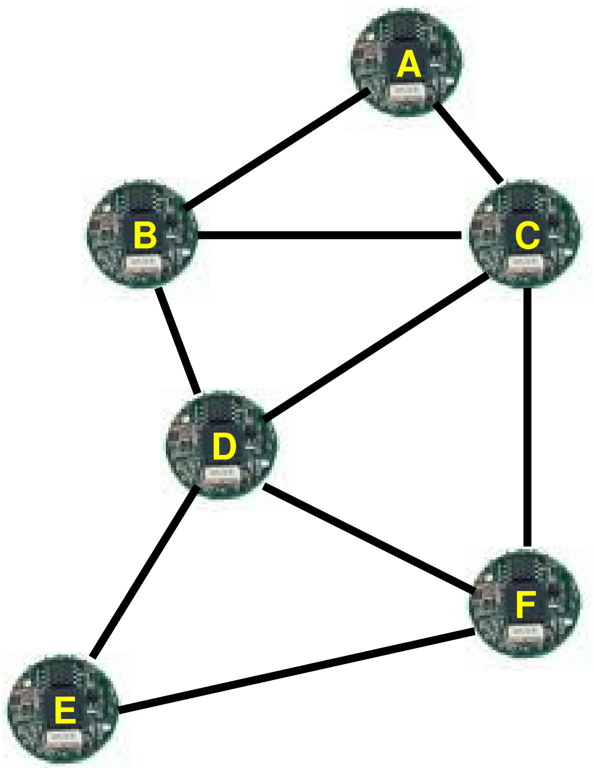
- Example:

$$\begin{aligned} AVG_{init}\{v\} &\rightarrow < v, 1 > \\ AVG_{merge}\{< S_1, C_1 >, < S_2, C_2 >\} &\rightarrow < S_1 + S_2, C_1 + C_2 > \\ AVG_{evaluate}\{< S_1, C_1 >\} &\rightarrow S_1/C_1 \end{aligned}$$

In-network Aggregation Example

- Example with the COUNT function:

```
SELECT COUNT(*)
     FROM sensors
```

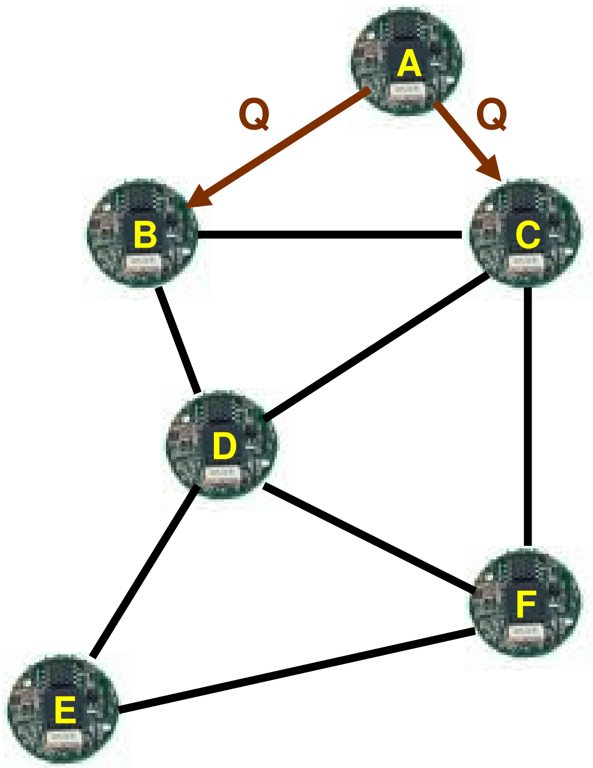


In-network Aggregation Example

- Example with the COUNT function:

```
SELECT COUNT(*)
    FROM sensors
```

- *A* queries its neighbors

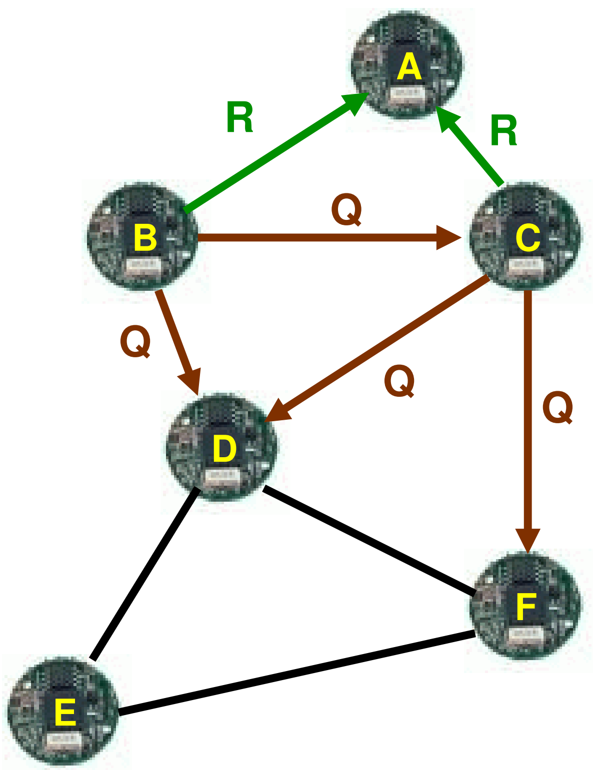


In-network Aggregation Example

- Example with the COUNT function:

```
SELECT COUNT(*)
    FROM sensors
```

- *A* queries its neighbors
- *B* and *C* respond and query their neighbors
- *A* believes COUNT is 3

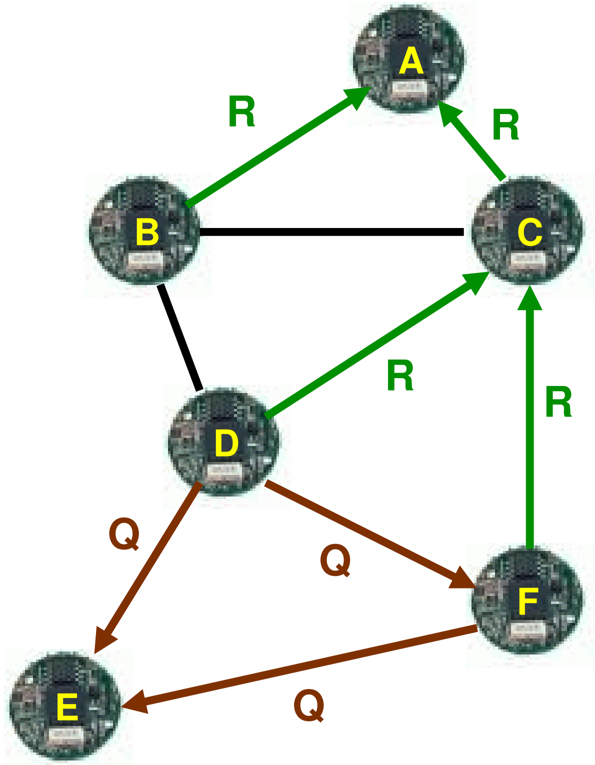


In-network Aggregation Example

- Example with the COUNT function:

```
SELECT COUNT(*)
    FROM sensors
```

- B and C query their neighbors
- D and F respond to just one parent
- A believes COUNT is 5

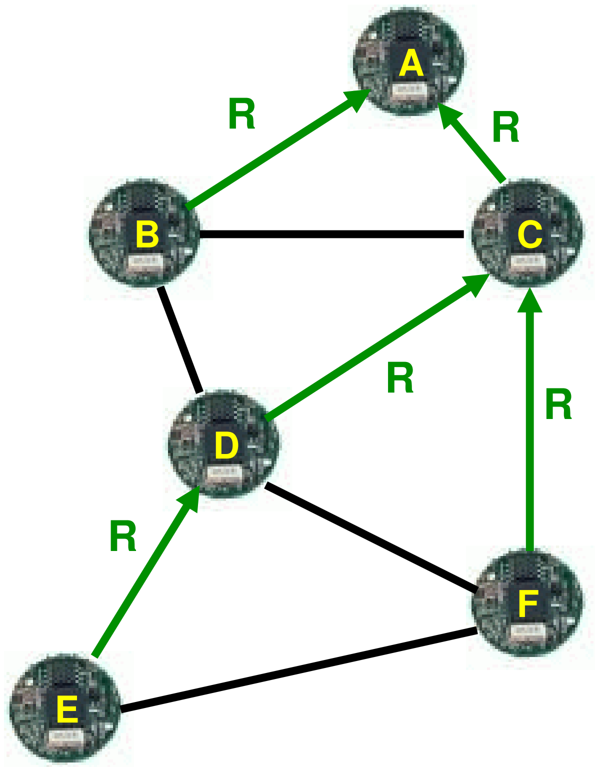


In-network Aggregation Example

- Example with the COUNT function:

```
SELECT COUNT(*)
    FROM sensors
```

- E starts responding to D
- Aggregation tree is fully deployed
- A believes COUNT is 6



Evaluation of TinyDB

- Advantages
  - Nice database abstraction on top of a generic sensor network operating system
  - Powerful programming abstraction
  - Aggregation functions are extensible
  - Actuators integrated in the operating system
- Limitations
  - Reconfiguration is not possible
  - Applications have no control over optimization parameters
  - Applications are required to provide most services

Virtual Machines

Features

- Provide the flexibility of a complete computing system in each sensor node
  - Flexibility of the virtual machine environment is important
  - Smart message, active message, mobile agent abstractions
- Energy considerations play a crucial role
- Overhead associated with running the virtual machine
- Mostly available for environments with hardware with more capacity (iPAQs vs. MICA2 motes)

Features in more Detail

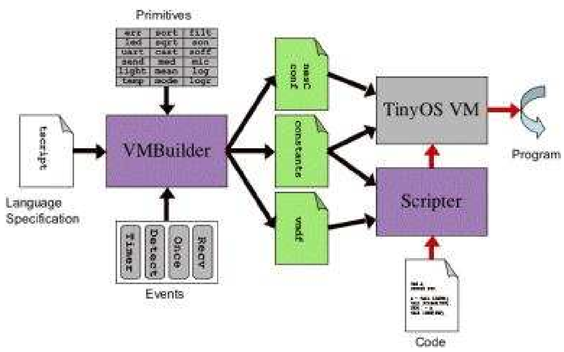
	Maté	Smart Messages	Agilla	SensorWare
Abstraction	program capsules	agents (messages), communication	mobile agents, tuplespaces	TCL-scripts, active sensor
Efficiency	energy	energy	energy	emergent, energy
Programmability	configurable at compilation	Java-based	based on Maté	TCL (with extensions)
Adaptability	code migration	message migration	agent migration	code migration
Scalability		Java		iPAQs
Topology		mobile ad-hoc	multi-hop	
Security	(planned)	trust, malicious SMs		
QoS		fault-tolerant		

The Maté Virtual Machine

- **Goal:** Small, efficient virtual machine implementation for sensor networks
- **Philosophy:** Efficient sensor reprogramming is best performed with capsules in a virtual machine
- **Methodology:**
  - Implemented on top of TinyOS
  - Based on Active Message technology
  - Viral solution to propagation of programs, which can be broken into capsules
  - Configurable virtual machine engine for the execution of capsules

Virtual Machine Configuration

- Virtual machine configuration
  - Selection of a language
  - Selection of events
  - Selection of primitives
- Generation of files
- Execution of programs and/or scripts



basic	`00iiiiii`	i = instruction
s-class	`01iiixxx`	i = instruction, x = argument
x-class	`1ixxxxxx`	i = instruction, x = argument

- 8 user-defined instructions

Code Execution

- Maté is a stack-based architecture
- It uses three execution contexts
  - Clock timers
  - Message receptions
  - Message send requests
- Each context has two stacks: operand and return address
- Operand types: values, sensor readings and messages
- Data sharing among contexts by means of a single shared variable

Code Execution Example

```
pushc 1      # Push one onto operand stack
add          # Add the one to the stored counter
copy         # Copy the new counter value
pushc 7
and          # Take the bottom 3 bits of copy
putled       # Set the LEDs to these three bits
halt
```

- Very simple program that just takes a value from the stack and sends it to the LEDs for visualization
- Series of instructions combined in capsules of up to 24 instructions

Code Capsules and Execution

- Every code capsule includes type and version information
- Four types of capsules:
  - Message send capsules
  - Message receive capsules
  - Timer capsules
  - Subroutine capsules
- Use of version numbers to implement code infection throughout the network
- Constrained execution environment helps take care of malicious capsules

Evaluation of Maté

- Advantages
  - Increased security by the use of a virtual machine
  - Code size reduced due to the use of common opcodes
  - Configurable virtual machine
  - Epidemic capsule distribution method
- Limitations
  - Energy consumption for long-running and/or complex applications is prohibitive
  - All applications must fit the defined instruction set
  - Run-time overhead due to virtual machine execution

Adaptive Middleware

Features

- Independently of the specific abstraction, adaptation plays a crucial role
  - Proactive adaptation allows the application to specify under which conditions should be adapted
  - Reactive adaptation monitors the system and reacts accordingly
- Cross-layer and, in general, optimization is key
- Scalability and security is normally not an issue for adaptive middleware solutions
- QoS and the ability to react to the environment are a common trend

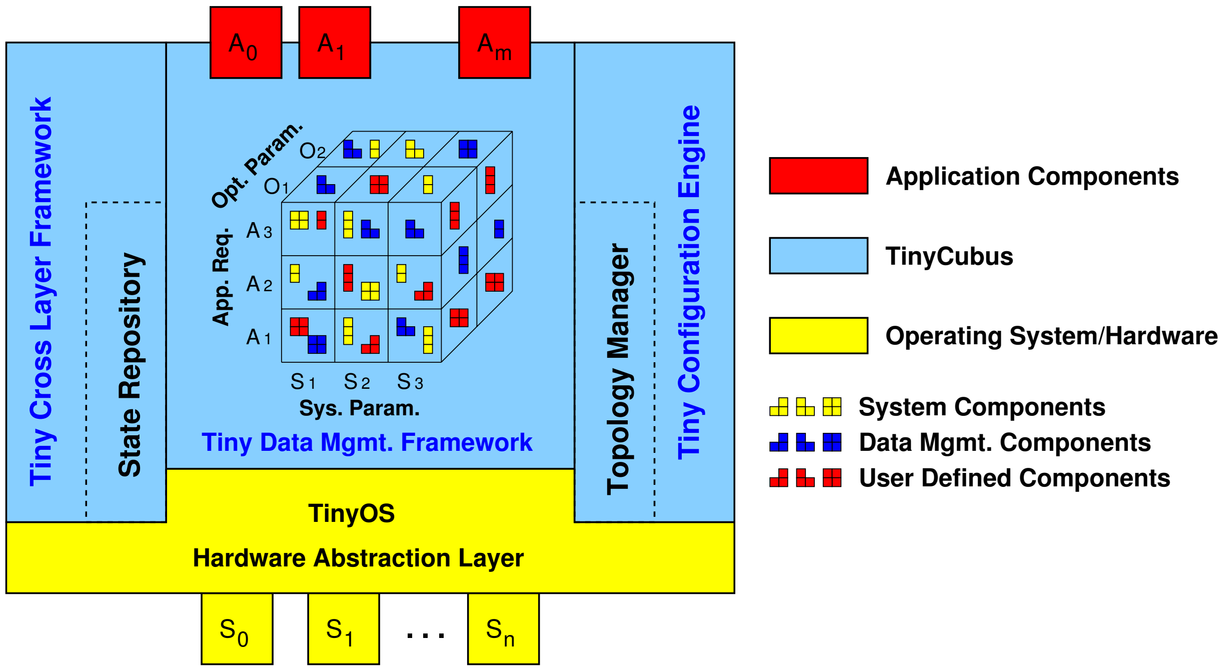
Features in more Detail

	MiLAN	AutoSeC	TinyCubus
Abstraction	communication (remote invocation)	dynamic service brokering	component-based system
Efficiency	energy-aware, cross-layer	cross-layer	optim. parameters, cross-layer
Programmability	image		component
Adaptability	proactive	proactive	proactive, reactive
Scalability			
Topology		infrastructure	ad-hoc, hybrid
Security			key exchange
QoS	fault-tolerance, QoS support	QoS	fault-tolerance, QoS

The TinyCubus Project

- **Goal:** Development of a generic reconfigurable system software for sensor networks
- **Philosophy:**
  - "Flexibility and adaptation are the key issues"
- **Methodology:**
  - Implementation on top of TinyOS
  - Definition of generic frameworks to allow for flexibility and adaptation
  - Provision of a set of standard components
    - System components
    - Data management and querying components

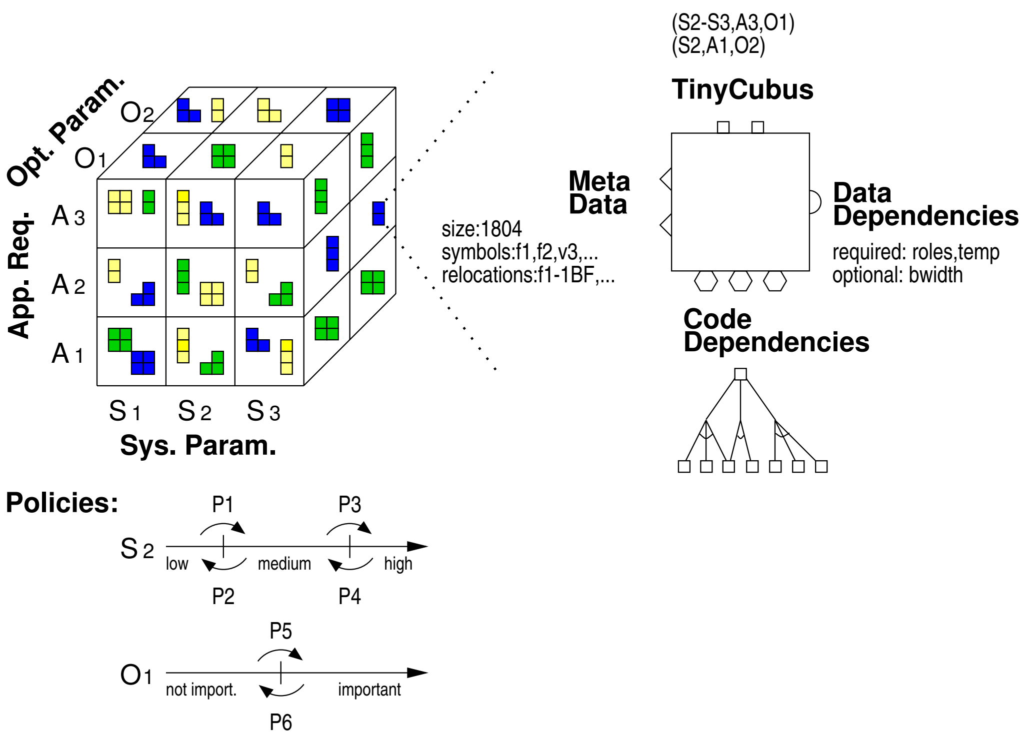
TinyCubus Architecture



Tiny Data Mgmt. Framework

- **Goal:** Provide a set of standard and adaptive data management components
- **Tasks:**
  - Choose the best set of components based on three dimensions:
    - System parameters: node density
    - Application requirements: consistency
    - Optimization parameters: energy, communication
  - Provide a set of system components such as time synchronization, broadcast strategies, etc.
  - Provide a set of data management components: replication, aggregation, consistency, etc.
- Adaptation and optimization strategies

Tiny Data Mgmt. Framework



Tiny Cross-Layer Framework

- **Goal:** Generic support for parameterization of components and applications
- **Tasks:**
  - Support for callbacks and/or user-level functions
  - *State repository* manages cross-layer data available from system and application components
  - Runtime support for cross-layer interactions
  - Distributed state management

Tiny Cross-Layer Framework

- Sample state repository

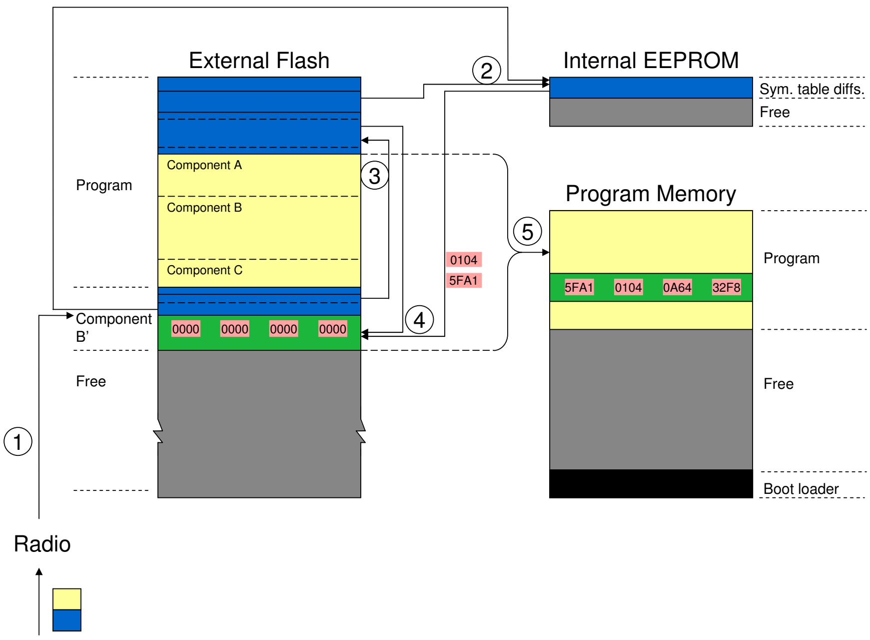
Name	Type	Publishers	Subscribers	Data
roles	I_{roles}	(system)	req:C3	n1={r1}
comp	I_{comp}	(system)	(system)	n1={C1,C2,C7}
pol	I_{pol}	(system)	(system)	n1=(S1,(10,27,35))
temp	float	C1,C5	req:C4,C5	n3=24.01
bwidth	int	C2	req:C5,opt:C3	(n1,n3)=42

Tiny Configuration Engine

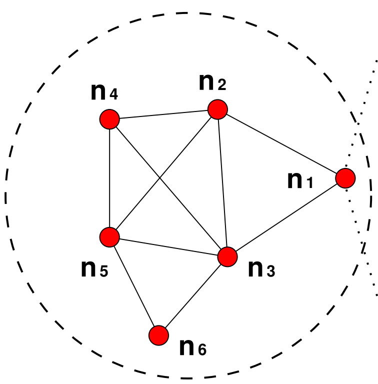
- **Goal:** Support for (re)configuration of system and application components
- **Tasks:**
  - Allows for the configuration/initialization of nodes using wireless technology
  - Determination of roles based on user specifications
  - Topology management
  - Encapsulation of access control policies for dynamic reconfiguration
  - Management of the current set of system and application components available at the sensor node

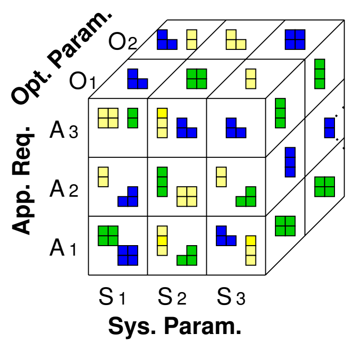
Tiny Configuration Engine

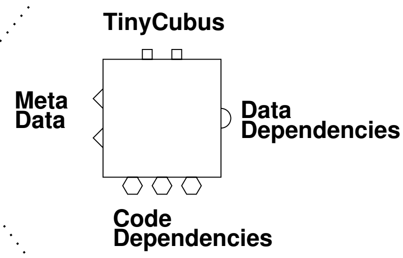
- (Re-)Configuration process



TinyCubus Integration







Name	Type	Publishers	Subscribers	Data
roles	I_{roles}	(system)	req:C3	n1={r1}
comp	I_{comp}	(system)	(system)	n1={C1,C2,C7}
pol	I_{pol}	(system)	(system)	n1=(S1,(10,27,35))
temp	float	C1,C5	req:C4,C5	n3=24.01
bwidth	int	C2	req:C5,opt:C3	(n1,n3)=42

Evaluation of TinyCubus

- Advantages
  - Flexibility allows it to be used in very different environments
  - Classification of components allows for efficient code selected both at compile-time and at runtime
  - Cross-layer support allows for application optimizations to take place
- Limitations
  - Overhead might be prohibitive in some environments
  - Adaptation policies are currently static
  - Scalability needs to be studied more closely

Comparison and Conclusions

- As usual, there are quite a few ways to solve the same problem
- Which type of middleware is optimal depends on:
  - Characteristics of the specific application at hand
  - Characteristics of the environment
  - Optimization criteria
- Adaptive middleware solutions offer some of the needed flexibility
- Sometimes the overhead is just not worth it
- Without adaptation, "if all you have is a hammer, everything looks like a nail"

Comparison and Conclusions

- There is still a lot of work to do:
  - Complex data processing
    - Multi-query optimizations
    - Operator placement
  - System architectures for data processing
  - Adaptation/optimization strategies
  - Streaming
  - Support for mobility
  - Hybrid network topologies
  - Miniaturization of sensors
- This poses many interesting challenges!

Thank You for Your Attention

Dr. Pedro José Marrón

University of Stuttgart
IPVS, Distributed Systems Group
Universitätsstr. 38
D-70569 Stuttgart
Germany

Phone: +49-711-7816-223
Fax: +49-711-7816-424

pedro.marron@informatik.uni-stuttgart.de

References

- Introduction:
  - Deborah Estrin, Ramesh Govindan, John Heidemann and Satish Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. *Proc. of MobiCom '99*, 1999
  - Ian F. Akyildiz, W. Su, Yogesh Sankarasubramaniam and Erdal Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, Vol. 38, No. 4, 2002
  - Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler and Kristofer Pister. System Architecture Directions for Networked Sensors. *Proc. of ASPLOS '00*, 2000
  - MICA2 Platform. `http://www.xbow.com`
  - Telos Platform. `http://www.moteiv.com`
  - BTNodes Platform. `http://www.btnode.ethz.ch`

References

- Middleware Challenges:
  - Kay Römer, Oliver Kasten and Friedemann Mattern. Middleware Challenges for Wireless Sensor Networks. *Mobile Computing and Communications Review*, Vol. 6, Nr. 2, 2002
  - Kirsten Terfloth and Jochen Shiller. Driving Forces behind Middleware Concepts for Wireless Sensor Networks. *Proc. of the REALWSN Workshop*, 2005
- “Classic” Middleware:
  - Ting Liu and Margaret Martonosi. Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. *ACM SIGPLAN*, 2003

References

- "Classic" Middleware:
  - Ting Liu, Christopher M. Sadler, Pei Zhang and Margaret Martonosi. Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet. ACM MobiSys, 2004
  - Carlo Curino, Matteo Giani, Marco Giorgetta and Alessandro Giusti. Tiny Lime: Bridging Mobile and Sensor Networks through Middleware. *Proc. PerCom 2005*, 2005
  - T. Abdelzaher, B. Blum et al. EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. *Proc. ICDCS 2004*, 2004

References

- "Classic" Middleware:
  - Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa and Carlos Ferraz. A Message-Oriented Middleware for Sensor Networks. *Proc. Workshop on Middleware for Pervasive and Ad-Hoc Computing*, 2004
  - Kamin Whitehouse, Cory Sharp, Eric Brewer and David Culler. Hood: A Neighborhood Abstraction for Sensor Networks. *Proc. MobiSys 2004*, 2004
- Data-centric Middleware:
  - Yong Yao and Johannes E. Gehrke. Query Processing in Sensor Networks. *Proc. of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, 2003

References

- Data-centric Middleware:
  - Philippe Bonnet, Johannes E. Gehrke and Praveen Seshadri. Querying the Physical World. *IEEE Personal Communications*, Vol. 7, No. 5, October 2000
  - Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. *SIGOPS Operating Systems Review*, Vol. 36, Nr. SI, 2002
  - Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong. The Design of an Acquisitional Query Processor for Sensor Networks. *Proc. of SIGMOD '03*, 2003
  - Shuoqi Li, Ying Lin, Sang Son, John Stankovic and Yuan Wei. Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. *Proc. IPSN 2003*, 2003

References

- Data-centric Middleware:
  - Shuoqi Li, Ying Lin, Sang Son, John Stankovic and Yuan Wei. Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. *Proc. IPSN 2003*, 2003
  - Chien-Chung Shen, Chavalit Srisathapornphat and Chaiporn Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, Vol. 8, Nr. 4, 2001
- Virtual Machines:
  - Philip Levis and David Culler. Maté: A Tiny Virtual Machine for Sensor Networks. *Proc. ASPLOS*, 2002

References

- Virtual Machines:
  - Porlin Kang, Cristian Borcea, Gang Xu, Akhilesh Saxena, Ulrich Kremer and Liviu Iftode. Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems. *Special Issue on Mobile and Pervasive Computing, the Computer Journal*, 2004
  - Chien-Liang Fok, Gruia-Catalin Roman and Chenyang Lu. Mobile Agent Middleware for Sensor Networks: An Application Case Study. *Proc IPSN'05*, 2005
  - Athanassios Boulis, Chih-Chieh Han and Mani B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. *Proc. MobiSys 2003*, 2003

References

- Adaptive Middleware:
  - A. Murphy and W. Heinzelman. MiLAN: Middleware Linking Applications and Networks. TR-795, University of Rochester, Computer Science, Nov. 2002
  - W. Heinzelman, A. Murphy, H. Carvalho and M. Perillo. Middleware to Support Sensor Network Applications. *IEEE Network Magazine Special Issue*, January 2004
  - Q. Han and N. Venkatasubramanian. AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering. *IEEE Distributed Systems Online*, Vol. 2, Nr. 7, 2001
  - Pedro José Marrón, Andreas Lachenmann, Daniel Minder, Jörg Hähner, Robert Sauter and Kurt Rothermel. TinyCubus: A Flexible and Adaptive Framework for Sensor Networks. *Proc. EWSN 2005*, 2005

References

- Adaptive Middleware:
  - Pedro José Marrón, Andreas Lachenmann, Daniel Minder, Matthias Gauger, Olga Saukh and Kurt Rothermel. Management and Configuration Issues for Sensor Networks.*International Journal of Network Management, Special Issue: Wireless Sensor Networks*, Vol. 15, Nr. 4, 2005
  - Pedro José Marrón, Daniel Minder, Andreas Lachenmann, Olga Saukh and Kurt Rothermel. Generic Model and Architecture for Cooperating Objects in Sensor Network Environments. *Proc. ICT 2005*, 2005