

RZ 3811 (# Z1111-002) 11/29/11
Electrical Engineering 21 pages

Research Report

Centrally Controlled Clustered Wireless Sensor Networks

Clemens Lombriser, Urs Hunkeler* and Hong Linh Truong


IBM Research – Zurich
8803 Rüschlikon
Switzerland

Contact author: Hong Linh Truong, email: hlt@zurich.ibm.com

*Now at: Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland

LIMITED DISTRIBUTION NOTICE

This report will be distributed outside of IBM up to one year after the IBM publication date.
Some reports are available at <http://domino.watson.ibm.com/library/Cyberdig.nsf/home>.

 **Research**
Almaden · Austin · Brazil · Cambridge · China · Haifa · India · Tokyo · Watson · Zurich

Centrally Controlled Clustered Wireless Sensor Networks

Clemens Lombriser, Urs Hunkeler, and Hong Linh Truong
 IBM Zurich Research Laboratory (ZRL)
 Rüschlikon, Switzerland

Abstract—We present *IMPERIA*, a centrally managed architecture for large-scale wireless sensor networks (WSN). Within the WSN, sensor nodes communicate using a clustered multi-hop TDMA protocol, which globally synchronizes the network and collects data at ultra-low power consumption.

The novel contributions to the state-of-the-art include a) an efficient algorithm for network topology discovery and link quality estimation, b) a clustering and routing algorithm for partitioning the complete WSN into multiple clusters with a-priori defined basestations, and c) a scheduling algorithm for multi-cluster and multi-channel data collection incorporating message aggregation.

Additionally, we discuss the advantages of using a centralized management over distributed approaches and present the design of a centrally managed energy-efficient WSN. The resultant architecture was deployed in four different applications and has been implemented for two embedded operating systems: TinyOS and IBM’s Mote Runner. We report on its performance and discuss opportunities for further research into the centralized approach.

I. INTRODUCTION

MOST wireless sensor network protocols today are designed for distributed decisions. Each sensor accesses the communication channel and communicates along a route it decides upon itself based on his local neighborhood. The common argumentation for this fact is that sensor nodes usually only have low computational resources such as a single microcontroller running at a few Megahertz and providing only a few Kilobytes of memory. Consequently they cannot maintain large network topology tables or run complex routing algorithms without seriously reducing resources available for applications.

Collecting the network topology at a single central entity to compute globally optimal solutions on the other hand is widely considered to be inefficient. There are reports stating that up to a whole third of the total available battery energy is used just to collect this information [1]. Additionally distributed solutions have been shown to be able to reach solutions at least close to optimal. Distributed solutions however have an overhead of control messages that has to the best of our knowledge never been compared to a well designed and optimized centralized solution. This overhead must be added in order to organize and adapt to new knowledge propagating through the network. Centralized solutions on the other hand can operate with little control overhead once they have been configured.

The most efficient way in terms of bandwidth and energy consumption for communication is to use time division multiple access (TDMA) protocols. They define exactly when

a sensor node needs to be actively sending or receiving and eliminate concurrent access to the shared communication medium. With TDMA, very low transceiver duty cycles can be reached while still maintaining high bandwidth. Achieving a low duty cycle is especially important for minimizing energy consumption; typical sensor nodes can consume three orders of magnitude less power when sleeping compared to when active. The IRIS mote for example draws just $12\mu A$ during sleep, but requires $10.31 mA$ when its transceiver is enabled. At low duty cycles, small additional power savings can significantly increase the sensor node lifetime [2].

The challenge is to establish a TDMA protocol that could span the whole WSN. Although distributed TDMA protocols have been published (e.g. Dozer [3]), they still include overhead for adaptation mechanisms in case collisions occur. Furthermore, very dense networks present additional problems as many sensor nodes content for the shared communication channel.

In this report we present the *Intelligent, Manageable, Power-Efficient and Reliable Internetworking Architecture* (IMPERIA). It is a centrally controlled architecture that vertically integrates a WSN network stack with the publish/subscribe messaging middleware MQTT-S [4]. The architecture can handle large-scale WSNs by forming clusters around multiple basestations. IMPERIA maintains global synchronization to run a network-wide multi-hop TDMA protocol, yet collects data to the local basestations for efficiency. The centrally executed algorithms for clustering, routing and scheduling make use of efficient mechanisms to discover the WSN topology and estimate link qualities. The energy spent during this initial phase is gained back by an ultra-low power operation for the remaining network lifetime, which usually is $> 99\%$ of the time a WSN is deployed.

The report is structured as follows: In Section II we describe common WSN application requirements and give an overview of the state of the art. In Section III we discuss the advantages of centralized WSN management over distributed solutions and present the principles we followed in designing IMPERIA. Section IV then details the different components of the architecture, while the following sections V and VI present the algorithms used for efficient network topology discovery as well as clustering, routing and scheduling. We discuss our experiences in implementing IMPERIA for two operating systems and in running four deployments in Section VII before we conclude the report. The appendix adds ideas for further work and goes into more detail on how the simulations

throughout the paper were created.

II. WSN APPLICATION REQUIREMENTS AND STATE OF THE ART

A. Applications Requirements

WSN applications cover a wide range of requirements [5], many of them being very application specific and requiring specialized protocols. Recently, it has been remarked that the theoretical requirements significantly differ from the actually implemented applications [6]. Here we summarize the most common characteristics that require non-trivial communication solutions.

Typical WSN applications include the deployment of a few dozen to thousands of sensor nodes. Such deployments are only economical if the size and in particular the cost of individual nodes is as low as possible as the cost of the network increases with the number of nodes deployed. A major cost advantage is using wireless instead of wired networks, which can reduce the total installation cost by 80% [7]. Wireless sensor nodes on the other hand need to be powered by batteries or some form of energy harvesting. In case they are battery-operated, sensor nodes should reach a long battery life time since a change of batteries is usually very costly or infeasible, in particular when the nodes are placed in remote area that are difficult to access.

The geographical area over which the nodes are spread is usually large compared to the range of the radio used for the wireless communication. Consequently, multi-hop networking schemes are needed. In a multi-hop network, nodes that are far away cannot reach the sink directly and have to send their data to intermediate relay nodes, which then forward it to the sink.

The data volume created by the nodes is usually very low, e.g. a few bytes per second, with the possibility of a significant larger volume in case of special events. A surveillance application for example may only report short status information messages during most of the time, yet deliver significantly more detailed and hence more voluminous sensor data when alarms occur. For such an application, it is also important that the alarm and its data are reported significantly faster than the status information, in some cases even in real-time. An additional problem is that due to a potentially large number of sensor nodes concurrently reporting an alarm, the aggregated data volume might exceed the capability of a single radio channel by far. In such a case, dividing the network into small subnetworks or clusters, each having their own radio channels may help meeting the application requirements.

In most applications sensor data is collected for "monitoring" purposes and is consolidated on machines running on legacy networks. Therefore the common communication of a WSN is preferably not point-to-point but tree-based, i.e. from and towards a sink located at the root of the routing tree. The data sent to wireless nodes usually contains management or control data and therefore is only of small volume.

From a network operator's perspective come additional, more practical requirements:

- **Ease of deployment** – The network should require as little configuration as possible. Ideally, sensor nodes are of just one type and are able to take different roles without configuration, such that an operator can place any node at any location without running into the danger that some might e.g. not support relaying functions. Additionally, the status of the sensor node and possibly its neighborhood should be testable right at the deployment. Other deployment considerations are discussed in [8].
- **Manageability and controllability** – The operator should at any time be able to quickly retrieve an overview on the network status. Errors or failures should not only be rapidly detectable, but also be reported in a way that easily allows them to be isolated and repaired. Especially with networks comprising a large number of nodes, it should be possible to quickly pinpoint the problem and take the appropriate action.

B. State of the art

A number of recently published WSN survey papers discuss the pros and cons of various architectures and protocols in great detail [9], [1], [10]. Here, we focus only on architectures based on TDMA, which is our preferred choice due to its inherent energy efficiency, bounded latency, fairness, and high throughput [1]. We further concentrate on centralized architectures, while distributed solutions are presented in [11],[3] or [12]. We believe that centralized architectures are superior to distributed ones in terms of scalability, controllability, and ease of management. The reasons are discussed in detail in the next section.

One of the most known TDMA-based protocol is the *Time Synchronized Mesh Protocol* (TSMP) [13]. TSMP is a pure TDMA MAC protocol, meaning that the network is always running in TDMA mode. Specialized slots are reserved for management purposes such as node discovery or link quality determination. An especially interesting feature of TSMP is how it establishes the synchronization required for TDMA. Common solutions use beaconing, but TSMP embeds time information into normal data and acknowledgment packets. TSMP assumes the existence of a centralized controller that coordinates and manages the TDMA schedules, it is however not known how this is done in detail.

The IMPERIA architecture in contrast uses TDMA only during its synchronized phase, during which it employs beaconing for synchronization to simplify the requirements on data link layer, which is recently being implemented in hardware for efficiency. Furthermore, this report on IMPERIA includes a complete description of the management procedures that are required for operating a WSN: from network discovery over routing tree construction and scheduling to clustering.

Another TDMA-based and centralized architecture is described in [14], in which a centralized gateway computes the TDMA schedule based on the battery levels of the sensor nodes and distributes the results to the nodes using pre-assigned slots. This architecture assumes that the gateway has an a-priori knowledge of the network topology and that all nodes are within the direct communication range of the

basestation. IMPERIA on the other hand details all required algorithms and can also work with sensor nodes located multiple hops away from the centralized base station.

G-MAC [15] divides the TDMA superframe into two periods, 1) a collection period during which the sensor nodes use a contention access scheme to send their traffic requirements to a basestation, followed by 2) a contention-free distribution period during which the sensor nodes exchange data messages with each other. The access schedules for the contention-free period are computed by the gateway based on the requirements received during the collection period and broadcasted by the basestation at the beginning of the contention-free period. G-MAC assumes that all nodes are within the communication range of each other.

PEDAMACS [16] also assumes that all nodes are within the broadcast range of the basestation. Similar to our architecture, PEDAMACS defines two operating phases: 1) a management phase during which the network topology is discovered, the TDMA schedule is computed and broadcasted to the nodes, and 2) a synchronized phase during which the nodes follow the TDMA schedule specifying when to send, receive, and sleep.

Our IMPERIA removes the need of having all sensor nodes within the communication reach of the basestation and allows them to be located multiple hops away. Further we use acknowledged unicast scheme during management mode to communicate between the basestation and the sensor nodes. This makes sure that the sensor nodes are correctly configured before we start synchronized phase. As will be shown later, our network discovery procedure is time bounded and more energy efficient than the distributed ones of PEDAMACS. We also provide a clustering algorithm which allows us to scale to networks with thousands of nodes.

III. A CASE FOR CENTRALIZED WSN MANAGEMENT

It is an almost dogmatic belief within the WSN community that centralized management of large networks has to be inefficient. We have encountered this statement in many publications however lack a reference proving the point. The belief in the efficiency of distributed solutions goes as far as that algorithms that do not allow a distributed implementation are not even accepted as reasonable solutions for WSN networking.

One of the main arguments quoted against centralized algorithms is that a global view of the network needs to be obtained before they can perform their work. Collecting this information is considered to scale badly and involve a substantial amount of communication and thus energy loss. It is however neglected that distributed algorithms initially require extra effort as well, and do need time until they settle into a stable configuration. We argue that with an efficient topology discovery mechanism the impact of establishing a global network is reduced far enough to make a centrally managed solution competitive. Such a mechanism is presented in Section V.

Another argument for distributed algorithms is that they are more flexible to adapt to changes in the network topology,

such as when nodes fail or link qualities change. Local decisions of the sensor nodes can quickly adapt and correct the problem, while loosing only little data that should be collected. A centralized algorithm however can as well take care of such situations and instruct the network to act accordingly. Additionally, research in link quality estimation has advanced, such that is better known how links of transient quality can be detected and thus avoided by routing protocols. In Section VII-F we report that within our laboratory, we reach a better performance with a stable network using a better link quality estimation than with a distributed approach adapting its collection parent over short times.

In many cases it is also possible and even desirable to detect bad connectivity already at deployment time, such that appropriate physical changes can be made and the WSN does not have to cope with too many bad links at all. Simply buffering messages over times of bad connectivity then may save a lot of algorithm complexity. We believe that it is generally better to fix bad connectivity with physical modifications at deployment time than to try to capitalize on short term improved links.

It is also often argued that a centralized architecture introduces a single point of failure, namely the basestation controlling the whole network. In many cases however it is the basestation that receives all data from the network, and if it fails, the WSN cannot export its data anyway, such that it becomes useless. In a centrally controlled approach, sensor nodes may be able to detect this situation much faster than a distributed protocol. It then can take appropriate measures such as going to a low-power waiting mode. If a distributed approach is badly designed, it may continue to run for a considerable amount of time while pointlessly wasting much energy in trying to find working basestations. A graceful degradation may be extremely valuable as a network-wide battery replacement may become necessary if the basestation fails.

A. Advantages over distributed solutions

We believe that centralized management of WSNs does have a number of specific advantages over distributed solutions:

- **Simplicity of the code on the nodes** – As the main intelligence of the network is running outside of the WSN, sensor nodes do not need to perform complex tasks. This requires less effort in programming and debugging, but also lowers the memory and processing requirements of the network stack. A lower complexity of the code also improves the predictability of the node behavior. Table I lists the functions we require a node to implement.
- **Observability and controllability** – Assuming that the gateway controlling the network provides more resources for storing and logging the network data as well as its decisions, it is easier to analyze problems a network experiences. This is especially important when the cause of network failures should be quickly determined in order to decide on appropriate actions.
- **Ease of management** – Changes to the network can be performed manually at the gateway. This is also a good point were expert knowledge can be introduced. In case

a wireless link e.g. looks good in its quality, but a human operator knows that it will fail for some reason that is not detectable by a network protocol, the operator can remove it manually from the network topology. Also routing and scheduling decisions can be overridden if they contain bad choices. As the centralized functionality can also be called from mobile nodes during deployment, it allows quickly checking connectivity and sensor node operation on site. Deployment reports have shown that this is an invaluable feature [8], [17].

- **Cost** – Initially, the WSN vision was that thousands of wireless sensors would be distributed in large amounts in the environment and would autonomously form networks. A fact that has been underestimated however, is that sensor nodes will never get as cheap as costing just a few dollars a piece, which would allow such applications. Rather, they will continue to cost at least about \$50+ – why? Considering that the costs for manufacturing and for software licenses will not disappear, it is difficult to set up business plans which actually allows a company to make money out of distributing millions of cheap sensors of which just a few sense meaningful data. Additionally, environmental considerations will limit the number of deployed sensor nodes especially in nature, where batteries should not be left back even in small quantities. Consequently, large networks will always cost several \$1000–10'000, at which most likely some training of the personnel using it will take place. Such operators will be able to take wireless link quality decisions during deployments and manage to the network behavior if they are given the means to do it.
- **Choosing network parameters** – WSN research has shown that there is no single optimal protocol for various types of WSNs. The performance of a network protocol is largely dependent on network density, the general topology (e.g. three dimensional vs. two dimensional, large asymmetries such as train or bridge networks, ...) [18], communication structure (single or multiple sink/source), position of the basestation within the network, bandwidth of the physical layer, etc. A centralized controller knowing the whole network topology can make better decisions on different network parameters than distributed algorithms which generally only use local information for decisions.
- **More general solution** – Last but not least, a centralized controller can locally run any distributed algorithm to determine a network configuration before deployment.

B. Our design principles

As discussed in the last section, we believe that for many applications, a centralized management offers advantages over a distributed solution. However, also the centralized WSN management must be well designed to perform efficiently. Here are the most important principles we followed when designing IMPERIA:

- **Only one transmitter at a time** – The basestation initiates all activities and only allows a single node to

Function	Description
Source routing	Forward a message along a source route written into the message.
Status reporting	Report battery voltage, configuration identifiers, and other status information.
Save configuration	Store information on how to behave in synchronized mode.
Acknowledging	Responding to a command using the reverse source route.
Periodical sleep	Periodically sleep in case no communication is occurring.
Topology discovery	
Neighbor discovery	Broadcast discovery messages, collect neighbor replies, and send back to caller.
Link probe (LP)	Broadcast a pattern of link probe messages.
Compute LP statistics	Compute LP statistics when receiving link probe messages.
Report LP statistics	Report the last collected LP statistics to the caller.
Synchronized mode	
Synchronizing	Receive a time beacon from the parent and synchronize to its time.
Schedule execution	Activate radio transceiver according to individual send/receive schedule.
Aggregate and buffer messages	Receives data from child nodes and buffer locally until next send slot.
Handle API	Handle commands and events of the IMPERIA API for the applications.

TABLE I
THE FUNCTIONS AN IMPERIA NODE NEEDS TO PROVIDE ARE RELATIVELY SIMPLE, YET ALLOW A VERY FLEXIBLE MANAGEMENT OF THE WSN. THE ONLY STATUS INFORMATION THAT NEEDS TO BE KEPT IS THE SCHEDULE INFORMATION FOR THE SYNCHRONIZED MODE AND TEMPORARILY THE REVERSE SOURCE ROUTE TO A COMMAND. NOTE SENSOR NODES ARE PASSIVE BY DEFAULT AND ONLY COMMUNICATE IF INSTRUCTED TO DO SO.

transmit at a given time. While this may seem terribly inefficient at the first glance, especially as there may be hundreds of nodes present in a network, it does solve quite a number of problems relating to the nature of wireless communications.

The main benefit is that the sender will find the channel free when it needs to use it. No or only simple contention protocols are needed. The only source of possible collisions comes from lost of acknowledgements, which may trigger a retransmission of a message while the receiver tries to forward it.

With this one transmitter at a time rule, messages travel quickly through the network to the destination node and back, such that the basestation can trigger subsequent activities at a high rate. Furthermore, the time duration of the activities are almost deterministic, thus easy to recover in cases of failures.

The only exception to this rule is the discovery procedure presented in Section V. After a broadcast of a discovery message, all neighbors attempt to respond at the same time. This is an exception for which to the best of our knowledge there is no efficient solution.

- **Management and ultra-low power synchronized mode of operation** – Initially, the wireless network is put into a management phase in which all nodes are in listening mode, waiting for commands sent by the basestation.

During this phase, they can be quickly accessed to execute procedures for topology discovery, synchronization and configuration. After having collected the topology information and configured the nodes for a TDMA schedule, we start the synchronized mode, in which the nodes spend the most of their time sleeping (i.e. radio transceivers off to save power). As a synchronized TDMA network allows to only turn on transceivers if there is actually something to send or receive, the sensor nodes run with minimal power consumption and may make up their initial leeway in energy consumption versus other low-power networks. For good results however, this requires the management phase to be brief and efficient.

- **Minimize amount of network information stored on the wireless nodes** – All network information we permanently store on a wireless node are its address and its TDMA schedule information. No additional data about neighbors or other network specific information, e.g. routing information, is kept. All messages exchanged between the basestation and the nodes during the management phase make use of source routing. During the synchronized mode the timing of the message identifies its target, such that it does not require a destination address at all.

IV. IMPERIA ARCHITECTURE

Based on the principles presented in the last section, we have designed the *Intelligent, Manageable, Power-Efficient and Reliable Internetworking Architecture* (IMPERIA) for Wireless Sensor Networks, which is shown in Figure 1.

IMPERIA organizes the WSN into one or multiple clusters depending on its size and topology. Clusters are created such that at least one permanently installed basestation with a connection to an *IMPERIA Gateway* (IGW) is found within them. The IGWs manage the sensor nodes within their cluster and collect the sensor data they generate. The decision of which sensor nodes belong to which cluster is taken by the *IMPERIA Global Controller* (IGC), which bases its decision on the network topology information received from the individual IGWs. The IGWs, the IGC, and client applications communicate using the open publish/subscribe messaging protocol MQTT-S [4], which is implemented by the IMPERIA broker. This simplifies scaling to larger networks with multiple IGWs and a large number of client applications. The number of connections increases only at the broker, which is specifically built for this task.

As many other TDMA-based architectures, IMPERIA operates in two modes: a *management mode* for network topology discovery and node configuration, and a *synchronized mode* within which data is collected from the sensor nodes. During the management mode, IMPERIA generally lets the sensor nodes remain in receive mode such that they can quickly respond to commands. Compared to other architectures, this practice will consume more energy during deployment and network topology discovery time. IMPERIA compensates for this additional energy usage by using an ultra-low-power TDMA protocol during the synchronized mode. Through the

centrally computed schedules, sensor nodes have only to wake up when they can send or receive data.

In the next sections, we discuss more details of the management and synchronized modes and present algorithms for their efficient execution.

A. Management mode – Topology Discovery and Configuration

Initially, wireless sensor nodes implementing the IMPERIA network stack are in receive mode. An IGW may issue commands to individual sensor nodes by sending them a message possibly over multiple hops using source routing. The sensor nodes execute the command and respond with the result by reversing the source route.

Within the management mode, the main tasks to be performed are the network topology discovery, the sensor node configuration, and the status checks. The first task includes discovering all the wireless links between the sensor nodes and their neighbors as well as measuring the quality of those links. The second procedure is needed to configure the sensor nodes for the synchronized mode and the last one is to verify this configuration as well as the correct operation of the node.

During the management mode, the IGC and the IGW ensure that always only one message is traveling through the WSN at any time. Sensor nodes respond with an acknowledgement message upon reception of a message at each hop it travels along the source route. In case the sending node does not receive an acknowledgement, it attempts to retransmit the message a number of times before dropping it. The sensor nodes use CSMA to send messages because the retransmissions of a first sender may collide with a transmission of the receiver over the next hop if an acknowledgment was lost.

Allowing only one transmission at a time may seem to be inefficient. We believe however that this rule improves the reliability and predictability of the communication through the network. The network still can be operated quite efficiently as we will show in Section V.

If multiple IGWs are used, the IGC may choose which one it will use to send messages to the sensor nodes. It may for example choose the one offering the shortest source route. For discovery and link probing, it will select one IGW to perform the discovery and link probing. The selected IGW will publish all results to the IMPERIA broker such that the other IGWs and the IGC receive them. It will also inform the IGC upon completion of its task, such that the IGC can continue issuing sequential commands.

In case that the source routing paths get too long or that the network seems not to be fully connected, the IGC may also choose to continue discovery or link probing from a different IGW.

The clustering, routing and scheduling decisions are generally executed by the IGC. If however no IGC is present or only a single IGW is connected to the network, the algorithms can also be run by the IGW itself.

The IMPERIA management mode includes a management sleep mode. If the wireless nodes do not receive any IMPERIA messages for an extended amount of time, e.g. 30 minutes or

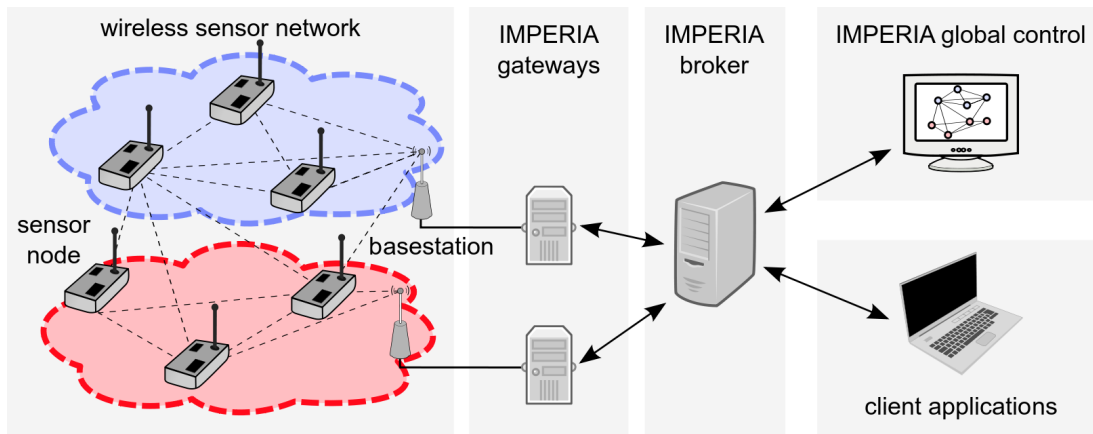


Fig. 1. IMPERIA architecture for end-to-end communication from the WSN to client applications. The architecture includes the wireless sensor network, one IMPERIA gateway per cluster, an IMPERIA broker to handle the communication between the gateways and the IMPERIA global control as well as to client applications.

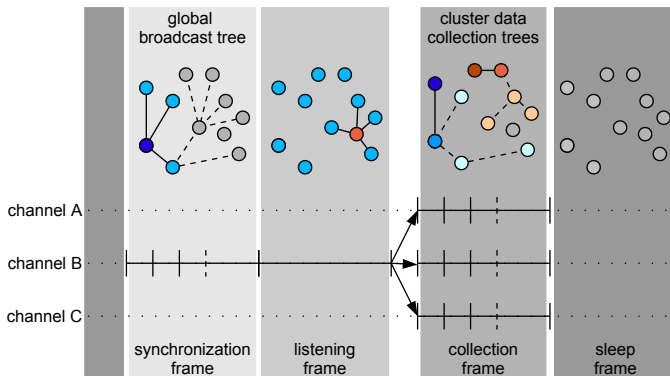


Fig. 2. The IMPERIA multi-channel TDMA superframe structure and the routing trees used during each frame. During the synchronization frame a single broadcast tree spans the whole network, while the collection frame uses individual unicast trees on different communication channels for each cluster. In the listening frame, new or un-synchronized nodes broadcast announcements to their neighbors only.

an hour, they start a low power management mode, during which they periodically wake up for a short time to listen for messages. If they do receive any IMPERIA message, they will remain awake for an extended period again. This saves energy during deployment time or in case synchronization got lost.

B. Synchronized mode – Clustered ultra-low power TDMA

During the synchronized mode, all wireless nodes within the WSN execute a TDMA schedule which defines when they have to be in receive mode and when they can send messages.

IMPERIA defines a superframe structure which is periodically repeated. The four frames constituting the IMPERIA superframe are illustrated in Figure 2 and defined as follows:

- 1) A *synchronization frame* within which synchronization beacons are forwarded along a network-wide broadcast tree. The function of the beacons is to establish synchronization between the nodes, such that they will start and end the subsequent TDMA slots at the same time. The broadcast tree is rooted at one of the basestations (the main basestation) and each parent node is assigned a

slot to broadcast its synchronization beacon to all of its children.

Sensor nodes that are in management mode can immediately enter the synchronized mode upon reception of a synchronization beacon from their parent and will then follow the schedule they are configured to follow. A sensor node expects to receive a synchronization beacon during each superframe. If synchronization beacons are missed for a number of times, the node will go back to the management mode (and eventually to management sleep mode). From there it may reenter synchronization mode immediately after the reception of the next synchronization beacon.

The synchronization beacons are sent within the same communication channel as used during management mode. This enables the sensor nodes to fall back into the known state of management mode, yet still be ready to join the network again. As the synchronization beacons themselves are short, the main IGW may decide on appending (piggy-backing) data messages destined for one or multiple nodes in the network. This allows a low-bandwidth data dissemination path intended for configuration data.

- 2) The second frame is the *listening frame*. This frame is optional and may be enabled by the broadcast tree root for one superframe to search for new or lost sensor nodes. All sensor nodes that are in synchronized mode will be in receiving mode on the management communication channel for the duration of the listening frame. The synchronization beacons include information on the start of the listening frame, such that sensor nodes which are still in management mode, but without valid configuration, may synchronize into the listening frame and broadcast an announcement message. All sensor nodes in synchronized mode which receive this announcement will forward it to the basestation within the next collection frame. That way, the gateway is notified of new nodes within the network, and how they are connected.

- 3) The third frame is the *collection frame*. Within this frame, sensor nodes forward their data along a collection tree to the cluster's basestation. Each cluster makes use of an individual communication channel, such that the data transmission can be performed in parallel. During this frame, there is within each cluster (and hence communication channel) always only one child node sending and one parent node receiving.
- 4) The last frame is the *sleep frame*, which covers the remaining time of the superframe. No communication takes place during this time and all sensor nodes may turn off their radio transceivers to save energy. Choosing the right duration of a superframe involves a trade-off between the frequency of sensor data delivery and energy consumption. What a good choice is depends on the application, but for maximum network lifetime, the sleep frame should consist as much as possible of the superframe duration. The minimum duration of a superframe on the other hand depends on the number of slots required for the broadcast and collection frames and the time reserved for the listening frame.

The synchronization beacons are a key element of the IMPERIA network stack. They contain flags for enabling optional frames, emptying collection data queues or requesting status reports. Additional fields include the time remaining until the next listening frame start, such that unsynchronized nodes can synchronize themselves into it, as well as the time until the next superframe will start and its total duration. Hence all timings are kept relative to the last synchronization beacon received, and no global reference time must be established.

An additional field of the synchronization beacon is the expiration field, which is set by the main basestation and allows to run a count-down until the synchronized mode should be left by all the nodes in the network. A time age field on the other hand is set by a parent node when it does not receive a beacon from its own parent. This field lets child nodes know that their parent is about to lose synchronization. Thus, if a parent node has not consecutively received a certain number of synchronization beacons, it will move to the management mode jointly with its whole subtree. Hence this field avoids having to wait for each hop to lose its synchronization individually.

In the following sections we will discuss the actual algorithms used within the management mode and for the configuration of the synchronized mode.

V. EFFICIENT TOPOLOGY DISCOVERY

A key performance measure for a centrally managed WSN architecture is how fast it can discover the topology of the WSN. The clustering, routing, and scheduling algorithms require information about the available links in the network and their quality to compute an efficient communication strategy for the synchronized mode. If the time for assessing the network topology is kept short, centralized protocols have better chances in competing with distributed solutions.

IMPERIA does use duty-cycling during the management mode, however only for extended periods where no communication occurs, such as at deployment time. If a sensor

Parameter	Value	Parameter	Value
$T_{deepsleep}$	10 s	T_{wait}	30 min
$T_{deepwake}$	50 ms	T_{send}	7 ms
m_{probe}	100 messages	$T_{maxbackoff}$	50 ms ¹
$m_{discovery}$	3 broadcasts		

TABLE II
PARAMETER VALUES USED TO COMPUTE THE NETWORK STARTUP TIME OF FIGURE 3

node in management mode does not receive any IMPERIA message for a long time (such as 30 min or an hour), it goes to management sleep mode. In management sleep mode, sensor nodes sleep for a time $T_{deepsleep}$ and periodically wake up and listen for a time $T_{deepwake}$.

To start up a network, all sensor nodes need first to be woken up from management sleep mode, then the network topology needs to be discovered and the link quality assessed. Finally, all sensor nodes need to be configured with their TDMA schedule before the main basestation can issue its time beacon to start synchronized mode.

In the following we present the algorithms used for the individual steps and at the same time compute the total startup time for a WSN of N nodes, which are deployed in a management sleep mode. We assume that the complete network is deployed when the process is started. Example values for the parameters we introduce are given in Table II.

- 1) **Wakeup** – An IGW starts the wakeup procedure by requesting its basestation to continuously broadcast a short wakeup message for $T_{deepsleep} + T_{deepwake}$. The sensor nodes listen for messages during their wakeup time $T_{deepwake}$. In case they determine communication on the channel, they wake up and attempt themselves to broadcast a wakeup message using CSMA for a time $T_{deepsleep} + T_{deepwake}$, then they remain listening. In case they do not receive any IMPERIA messages for an extended time T_{wait} , they go back to deep sleep. The time required for this wakeup procedure depends on the sleep/wake durations as well as the depth of the network in hops h , and has a maximum duration of

$$T_{wakeup} = (T_{deepsleep} + T_{deepwake}) \cdot h \quad (1)$$

After issuing the wakeup command, the IGW has to wait until the broadcasts have died down at least in its vicinity. It must therefore have an estimation of the maximal hop count of the network and hence also the maximum time it needs to wake up. After that, the IGW can continue with the actual network topology discovery. As an alternative to the wakeup algorithm, the IGW may also issue multiple rounds of discovery until all sensor nodes have been found.

- 2) **Link probe and discovery** – The second step is the discovery of all available links within the network and their quality. This process is controlled by the IGW, which ensures that at each point in time, only one measurement is performed. Interference effects from the

¹With standard IEEE 802.15.4 settings, the maximal backoff time plus message transmission is 38 ms [19], we add an additional 12 ms safety margin.

own protocol are minimized that way at the cost of an acceptable longer duration of the whole process.

When a node is instructed by the IGW to discover its neighbors, it first broadcasts a pattern of m_{probe} link probe messages. The neighboring nodes receiving these link probe messages record the number of link probe messages received and associated link quality measures such as RSSI, LQI, etc. After having sent the link probe pattern, the current node broadcasts a neighbor discovery message, to which the neighbors reply while adding the previously obtained link quality measurement results. After broadcasting a first neighbor discovery message, the current node waits for the maximum CSMA contention time $T_{maxbackoff}$ and rebroadcasts the neighbor discovery message with a list of nodes of which it has already received a reply. In this second round, only sensor nodes that are *not* included in the list reply to the broadcast. Including the replied nodes in the discovery message allows reducing the number of contenders for a reply and thus increase the probability to find all neighbors, especially within dense networks. The discovery rounds are repeated until no more nodes answer or for a maximum number of iterations $m_{discovery}$. All neighbor and link probe information is then returned to the IGW by the sensor node that had performed the discovery². The IGW keeps a list of nodes that have not yet performed a link probe and neighbor discovery, and instructs node by node to do it until the list is empty. The nodes are reachable via source routing, and the IGW can determine the best path at each iteration based on the already discovered links.

This assumes that the links are bidirectional to a certain extend, i.e. that messages can be exchanged in both ways with a certain probability. It has been found that links of high quality in one direction have at least an acceptable quality in the other, which would suffice for this procedure [20]. Links with low quality are also of reduced interest as they will likely not be used for routing.

The time $T_{discovery}$ spend by each of the N nodes to scan its neighborhood consists of sending the message via source routing to the node, requiring an expected number of transmissions along a route R $E[R]$, performing the link probe and discovery, then sending the message via reversed source route back to the basestation:

$$T_{discovery} = E[R] \cdot T_{send} + m_{probe} \cdot T_{send} + m_{discovery} \cdot T_{maxbackoff} + E[R] \cdot T_{send} \quad (2)$$

Note that the total time required for discovery is $N \cdot T_{discovery}$ and scales almost linearly. The component that increases faster than linear is the time required for

the source routed message to reach the selected node and return with the result. This time remains however small for practical network sizes when compared to the other terms.

- 3) **Network Configuration** – When the IGW has completed the network topology discovery, it can compute the routing and schedule for the network, and distribute the resulting configuration data to the sensor nodes. Assuming the availability of an efficient algorithm that completes in neglectable time (see Section VI for such an algorithm), the network needs to spend an additional $N \cdot E[R] \cdot T_{send}$ to distribute the network configuration messages to all nodes.
- 4) **Network start** – Having configured the nodes, the network can be started. This is done by the main basestation broadcasting a first synchronization beacon. Each sensor node receiving this beacon can enter synchronized mode and will issue synchronization beacons within its appropriate slot within the broadcast frame. The time until all sensor nodes have entered synchronized mode is thus equal to the duration of the broadcast frame. An evaluation of the broadcast frame duration Section VI-C shows that it is usually neglectable in comparison to the time required for discovery and link probe.

The duration of the complete startup process depends on three parameters that are only determined through the actual network topology. These are the number of nodes N , the maximum number of hops h and the expected mean number of transmissions to reach a node $E[R]$, which is the sum of the expected number of retransmissions required to send a message over each link of the route from the basestation to the node³:

$$E[R] = \frac{1}{|R_{all}|} \sum_{R \in R_{all}} \sum_{r \in R} \frac{1}{p(r)} \quad (3)$$

where R_{all} are all the routes from the basestation to the nodes, R is one of these routes, and r is an individual link between two nodes along the route R . The Packet Reception Rate (PRR) $p(r)$ finally indicates the probability that the transmission of a message over a link is successful.

An estimation of the total network startup time results from summing up the wakeup time T_{wakeup} , the topology discovery time $N \cdot T_{discovery}$ and the configuration time, in which a configuration message is sent to each sensor node $N \cdot E[R] \cdot T_{send}$. The time to compute the clustering, routing, and scheduling is neglected here.

Figure 3 shows the resulting total network startup time for networks of varying size. The results are obtained from simulations that are performed as discussed in Appendix A. Networks with the shown average hopcount $E[h]$ were simulated. For each network size, 1000 topologies were generated and a randomly chosen node was selected to be the basestation

³Note that this is only an approximation. Individual nodes will only attempt retransmission for a number of times, and the basestation will reattempt for a number of times if no answer has been received. The approximation is however sufficiently close when the source route includes only links with sufficient PRR (e.g. >80 %).

²Dense networks may require to send neighbor and link quality information using multiple messages to the gateway. In such a case the modified version of Appendix B may improve reliability.

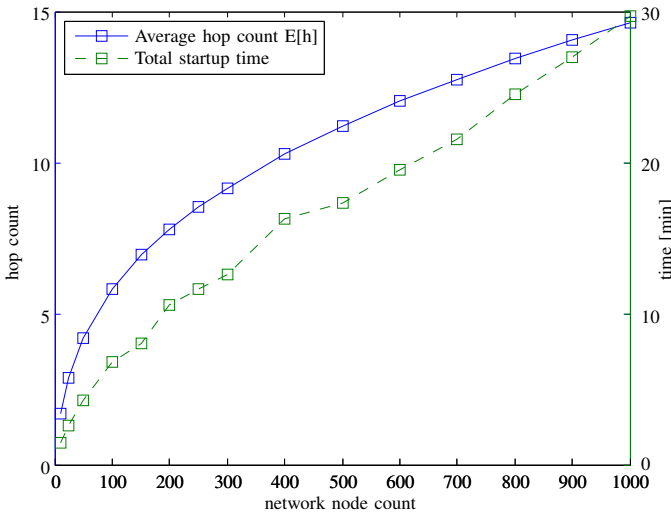


Fig. 3. Average number of hops and total startup time for varying network sizes

from which the network discovery was initiated. The resulting values show that networks of 300 nodes can be expected to have been discovered and started within 11 min while 1000 nodes require about 30 min. After the startup time, the networks will run in synchronized mode and thus highly optimized. We believe that this value is competitive compared to other decentralized low-power networks, which also need time to converge and stabilize on a given route and have additional overhead during this time. It is however difficult to find values in literature for a good comparison.

VI. CLUSTERING, ROUTING, AND SCHEDULING

If multiple IGWs/basestations are available within the network, the WSN can be partitioned into multiple clusters of sensor nodes which send their data to their cluster basestation. Clustering allows balancing the traffic load more evenly over the network, which reduces the overall time required to collect all sensor data, and additionally should allow reducing the duty cycles of the sensor nodes, especially the ones close to the basestations which have to forward the data of all nodes further away.

For our algorithms, we assume a number of IGWs connected to a number of fixed basestations that are distributed within the WSN, see Figure 1. Note that this differs from many approaches in the literature, where clustering is commonly used to locally aggregate data at a cluster head within the WSN [21]. This cluster head then forwards the data of its cluster to the closest basestation. Our IMPERIA architecture only allows basestations to be cluster heads⁴.

The clustering, routing, and scheduling processes are run at the IGC after it has obtained a global view of the WSN topology from the IGWs as described in Section V.

⁴In IMPERIA a base station is a wireless node that has a wired connection (e.g. via a serial port) to an IGW. Otherwise, there is no difference between a basestation and a normal wireless node.

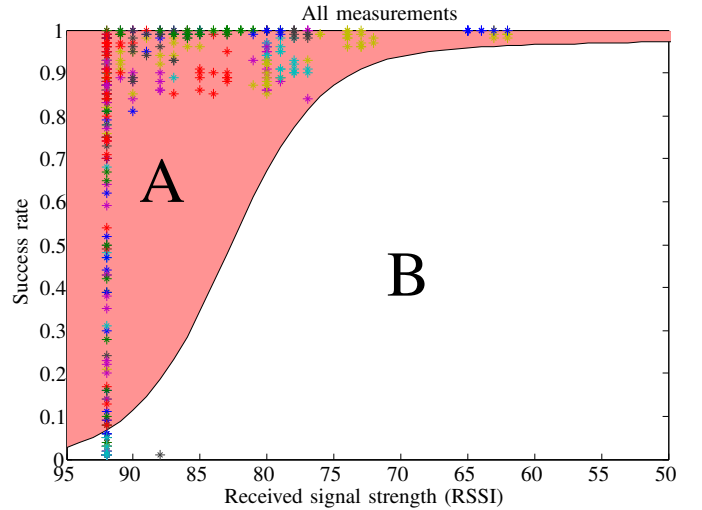


Fig. 4. A-priori distribution of the packet transmission success rate with the good link cut-off $m_{rssi} = -83$ dBm, the RSSI range $r_{rssi} = -91$ dBm, the transient link stretch coefficient $\alpha = \frac{1}{7}$, and the linear coefficient $\beta = 0.05$. It is multiplied with the measured success rates $p_{fwd}(r)$ and $p_{bck}(r)$ before using it as link quality indicator. Note that measured values in area A seem “to good to be true” and are corrected to lower values by the multiplication.

A. The DIVIDE clustering algorithm

The clustering algorithm used in IMPERIA is called *DIVIDE – Deep Internetworking via Independent Datasource Exploitation*. It is a combined clustering and routing algorithm that partitions sensor nodes into clusters such that their expected number of transmissions (EXP) to the basestation of the cluster is minimized. A low number of transmissions not only translates into a low number of TDMA slots needed for delivering the sensor data, but also helps saving energy.

The basis for the calculation of the EXP is the Packet Reception Rate (PRR) of both the forward and backward paths of a link [22]. The probability

$$p(r) = p_{fwd}(r) \cdot p_{bck}(r) \quad (4)$$

of a message being transmitted successfully over a link r is equal to the forward PRR ($p_{fwd}(r)$) of that link multiplied by its backward PRR ($p_{bck}(r)$) (for a successful return of an acknowledgment). The EXP of link r is then the expected number of transmissions required until a message has been transferred over that link, i.e.

$$E[r] = \frac{1}{p(r)} \quad (5)$$

A simple PRR measurement however is not enough to measure the quality of a link. Measuring link qualities has shown to be difficult and important enough that it has developed into an own area of research within the WSN community [20]. In general, the quality of a link can be classified into connected, transient, or disconnected. A connected link is a link with constantly high PRR, while a disconnected link has constantly low PRR. The transient links are the ones responsible for one of the main challenges in WSN research. Those links may temporarily have very high or very low PRRs, and may vary with environmental conditions. While many WSN

protocols attempt to use such links by allowing temporary reconfiguration, we aim at avoiding them in favor of better, if possible connected links.

An additional metric to identify the link quality is the Received Signal Strength Indicator (RSSI) which has been shown to have a quite predictable relation to the PRR for frame-based transceiver [23]. Measurements suggest that thresholds on RSSI can be defined above which links are connected. A similar value is the Signal to Noise Ratio (SNR) [20], which has an even better prediction value, but is not as easily accessible on most transceivers.

Equation 6 shows how IMPERIA estimates the PRRs based on measured PRR and RSSI values. The multiplication attempts to adapt the measured values closer to an a-priori known relation between RSSI and PRR. The function of Equation 6 is shown in Figure 4 and compared to RSSI and PRR values measured with the AT86RF230 transceiver on an IRIS mode. Note that the minimum RSSI value reported by the AT86RF230 seems to cover most of the links with reception rate of $< 80\%$, such that links in the transient range cannot be easily distinguished. Other receivers such as the CC2420 have an RSSI range that ranges over links with even lower PRR.

The multiplication with Equation 6 penalizes links that look better than expected (region A), but maintains the measured success rate for links with bad performance despite a good RSSI. When using the corrected PRR for computing the EXP, links with stronger RSSI are preferred. The equation has three parameters which need to be adapted to the transceiver used. These are the cut-off RSSI for good links m_{rssi} , a transient zone stretch factor α , and a linear coefficient β which favors links with better RSSI in the connected region. Besides the measured PRR $p_{fwd}(r)$ also the measured RSSI in the direction of the link $rssi(r)$ is used for the calculation. Note that the backwards PRR $\hat{p}_{bck}(r)$ is computed in an analogous way.

From the corrected transmission probability, the clustering algorithm computes the cost to each basestation and assigns each sensor node to the best cluster c as follows:

$$\hat{p}_{fwd}(r) = \frac{1-\beta}{2} \tanh(\alpha(rssi(r) - m_{rssi})) \cdot p_{fwd}(r) + \beta \left(\frac{1}{r_{RSSI}} rssi(r) + \frac{1}{2} \right) \quad (6)$$

$$E[R(i, c)] = \sum_{r \in R(i, c)} \frac{1}{\hat{p}_{fwd}(r) \cdot \hat{p}_{bck}(r)} \quad (7)$$

$$E[R_i] = \min_c E[R(i, c)] \quad (8)$$

where $R(i, c)$ is the route from node i to the basestation of cluster c , and each hop $r \in R$ has a transmission probability of $p(r)$. $E[R(i, c)]$ represents thus the corrected EXP required to complete the route from node i to the basestation of cluster c , assuming infinite retransmission attempts and no buffer overruns. The route with the minimal corrected EXP is R_i . It identifies both the cluster to which node i belongs and the route from node i to that cluster's basestation.

Other approaches for routing decisions, e.g. Arbutus[24], also exploit the Link Quality Indicator (LQI) values. LQI

is however not standardized and may be implemented in different ways by transceiver manufacturers. Additionally, Srinivasan [23] also report that LQI has a larger variance than RSSI for the same success rate on the CC2420.

A method for implementing the DIVIDE clustering approach is shown in Algorithm 1. It implements a breadth-first search from each basestation and assigns the minimum corrected EXP found for each node and the parent to which to forward the packets. For this purpose it initializes `curLevelNodes` with the list of basestations. It then iterates through the nodes in `curLevelNodes` and checks whether each neighbor could be reached with lower corrected EXP or with better cluster balance from the current node. If this is the case, the parent and current corrected EXP cost are updated, and the neighbor is added to the `nextLevelNodes`, the list of nodes to be evaluated next.

Algorithm 1 The DIVIDE clustering algorithm

```

1: curLevelNodes = basestations
2: exp(1 to nodecount) = infinity;
3: parent(1 to nodecount) = -1;
4:
5: while curLevelNodes not empty do
6:   for all node in curLevelNodes do
7:     for all neighbor of node.neighbors do
8:
9:       nextExp = exp(node) +  $\frac{1}{p_{corr}(neighbor \rightarrow node)}$ 
10:
11:     if nextExp < exp(neighbor) OR
12:       (nextExp == exp(neighbor) AND
13:        ccount(neighbor) < ccount(node)) then
14:
15:       exp(neighbor) = nextExp
16:       parent(neighbor) = node
17:       nextLevelNodes.add(neighbor)
18:     end if
19:   end for
20: end for
21:
22:   curLevelNodes = nextLevelNodes
23:
24: end while

```

The decision on line 13 determines whether a parent (and with it the cluster) is changed if the EXP is lower, or if the EXP is equal and the neighbor cluster is smaller than the current node's cluster. The cluster node count function `ccount` is not listed in more detail for the sake of brevity, but can easily be implemented using additional variables and arrays for keeping track of the nodes' current cluster assignment and the number of nodes in each cluster.

A useful byproduct of this clustering algorithm is that it provides a routing path for collecting from the sensor nodes to the cluster controllers. This routing can be used to schedule the transmissions for the collection frame of IMPERIA's synchronized mode. Algorithm 1 can also be used to compute the routing tree for the broadcast frame by initializing `curLevelNodes` with the basestation of the

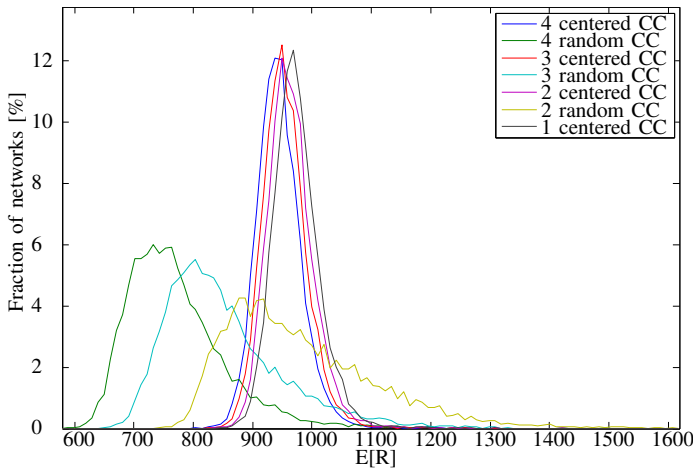


Fig. 5. The expected number of transmissions required for a network of 200 nodes when using clustering. The position of the basestations has a strong influence over the number of required transmissions for the network.

main IGW to be used as tree root and reversing the direction of the link transmission rate $p_{corr}(node \rightarrow neighbor)$ on line 9. Note that the EXPs for the broadcast frame only includes the forward PRRs as broadcasts are not acknowledged.

To evaluate the benefit of using clustering, we have generated 20'000 random network topologies with 200 nodes and 1 to 4 basestations. The basestations were either positioned in the center of the area or randomly selected from the nodes. The reason for the centrally positioned basestations is to evaluate the benefit of just parallelizing the data transmissions. Some applications may place restrictions on the positions of the basestations and only allow to use multiple collocated basestations on parallel channels. Details on how the topologies were generated are given in Appendix A.

Figure 5 shows a probability density function for the expected total number of transmissions needed for data collection $E[R]$. A single basestation in the center will introduce an expected total number of 980 (100%) transmissions. For four randomly placed basestations 771 (79%) transmissions are needed, for three 851 (87%), and for two 946 (97%). For centered basestations the values are 956 (98%), 965 (98%), and 975 (99%), respectively.

The conclusion of these results is that using multiple clusters does reduce $E[R]$ substantially if the basestations are not collocated. The long tails of the distributions are a result of the additional basestations being located in regions of the network which connect to the rest through an area to which another basestation has shorter access. If for example all basestations are located within the south-eastern corner of a WSN, it is likely that the basestation that located the most north-west will have to serve a large part of the network and the additional basestations serve only small parts of the network. A special case are the centrally placed basestations, whose benefits lay merely in allowing multiple communication channels to be used in parallel, hence speeding up the communication. $E[R]$ is however not significantly reduced. If basestations are more favorably distributed however, the transmission count is considerably reduced.

IEEE 802.15.4 MAC frame		
header	payload	CRC
≥ 9 octets	IMPERIA messages	2 octets

Length	Type	NodeID	Timestamp	Data
1 octet	1 octet	2 octets	2 octets	(Length-4) octets

TABLE III
IMPERIA MESSAGE FORMAT

B. Message Format and Aggregation

An additional measure for reducing the number of transmissions within the network is to introduce message aggregation. Within IMPERIA, short messages are concatenated such that in an extreme case a data collection tree parent can send its own and all of its children messages with a single message to its parent.

Transmitting a single large message requires less time than transmitting multiple smaller messages and is therefore more energy efficient. A minimal transmission over IEEE 802.15.4 includes a 4-octet preamble, 1-octet start field, 1-octet length field, at least 3-octet header data and a 2-octet checksum. At 250 Kbps, this takes $352 \mu s$. Each additional octet adds another $32 \mu s$. After transmitting a frame, the transceiver needs to switch from send to receive mode to receive an acknowledgment, which takes $192 \mu s$ according to the IEEE 802.15.4 specification. The acknowledgment itself has minimal transmission time, after which the transceiver has to switch back to send mode again before being able to send the next message. Consequently, sending 5 data messages with 10 bytes individually takes theoretically $5 \cdot 1'408 \mu s = 7'040 \mu s$ whereas it takes only $2'688 \mu s$ when they are sent within a single frame. This is 62% faster if the minimum timings are considered and will be even more when considering that messages need to be transferred from the radio transceiver to the micro-controller and back for buffering between TDMA slots.

The format of the IMPERIA messages used within the WSN is designed such as to allow easy aggregation and is shown in Table III. Each message begins with a *Length* field indicating the total length of the message including the *Length* field⁵ itself. The *Type* field indicates the unique type of the message, which identifies how the data should be interpreted. The values 0–127 are reserved for the IMPERIA network stack, while values 128–255 indicate application data which will be published as MQTT-S topic identifier, see Section VII-A. The third field of the message is a *NodeID* which identifies the source node address in case the data travels from the network node to the basestation, and the destination node address in case it travels from the basestation to the network node.

The last header field of the message is a *Timestamp* that is given relative to the beginning of the current superframe. In case a separate timestamp message is found within the same IEEE 802.15.4 frame, the header timestamp is given relative

⁵In case a link allows for payloads larger than 256 bytes, the *Length* field can be extended by setting the *Length* field to 0x01. In this case, 2 additional octets will be added to the *Length* field that will indicate a message length of up to 64 kB.

to the preceding timestamp. This may be used for messages that are buffered between superframes. Timestamp messages containing a full-scale timestamp are updated at each hop to the local sensor node time domain and converted into a globally valid time at the basestation.

At the sensor nodes, incoming IMPERIA messages may be aggregated by simply appending them to the last received or locally generated message. All messages are then sent following the IEEE 802.15.4 frame header. When determining whether there is an additional message appended, the total length of the frame can be used, or alternatively a Length field value of zero can be inserted.

The IMPERIA network stack also uses the message format shown in Table III. It adds source routes as an individual message and defines all subsequent messages to be destined to the node identified by the source route⁶. This allows sending multiple commands to a sensor node at once. During the synchronization frame, the aggregation allows piggy-backing additional data for all or specific nodes into the synchronization beacons. An interesting observation that can be made here is that the IEEE 802.15.4 header does not have to include any addressing information as during the management mode frames are either broadcast or include a source route identifying a destination⁷, and during synchronized mode frames can only be received by the intended destination.

Note that using this scheme would theoretically allow sending aggregated frames with multiple source routes and messages. If the source routes share the first few hops the frame could be broadcasted to two receivers that are the first hops where the source routes differ. This would exploit the broadcast nature of the wireless link to optimize transmission along a number of hops. This practice would however violate our principle of having only one sender at a time and would introduce more internal protocol interference⁸.

Many sensor network applications only require the collection of a few bytes of data per sample. Typically, each sensor node generates one message per superframe, which will be forwarded through the network to the basestation. If each parent aggregates its own and its children messages, it can reduce the number of send slots it needs for sending its data to its parent, and hence save energy not only for itself but also for its parent.

The benefit of using aggregation is shown in Figure 6. It shows two histograms with the number of nodes having to transmit a certain number of frames. In Figure 6a no aggregation takes place, every received message is forwarded as it is. In Figure 6b however, up to 10 messages are aggregated into a single frame for the transmission. The figures show the average results from the simulation of 10'000 networks of varying size.

⁶Note that by adding the source route as a variable-length message, the maximum route length is limited by the maximum payload of the IEEE 802.15.4 frame and by the length of the subsequent messages that follow.

⁷The source route also includes the current position, such that a node overhearing a frame that comes later in the source route cannot confuse it with its own turn to forward the frame.

⁸Of course, CSMA could be used to manage concurrent wireless access, but it would introduce a need for decisions on which types of commands can be executed in parallel and would make the whole communication less deterministic.

Let's take a network of 500 nodes as a comparison point. As can be seen in Figure 6a there is on average 1 node having to transmit a maximum of 35 messages without aggregation. With aggregation on the other hand, the maximum for 1 node lies at just 20 frames to be transmitted.

The ripple effect seen in Figure 6b results from the aggregation. A sensor node is more likely to have an even number of send and receive operations with aggregation. After a receive, it can attach its own message to the frame and forward it in a single send operation instead of having to add another send.

C. Scheduling

During the IMPERIA synchronized mode, each sensor node follows its own schedule and only activates its transceivers for receiving or sending messages within its assigned slots. Send and receive slots are assigned such that the broadcast and data collection routing trees deliver the data to the sensor nodes and the basestations respectively. Note that the sending time uniquely identifies the receiver of a message, hence they do not have to include any addressing.

The goal of a scheduling algorithm is to find a schedule with the minimal number of slots required to transfer the data along a given broadcast or collection tree. To be optimal, the scheduling algorithm needs to know the sizes of the messages each node in the network will send. Additionally, it requires knowledge about the size of the message buffers of all nodes, such that forwarding nodes are guaranteed to be able to buffer the received messages until they can transmit them. Finally, the scheduling algorithm needs to know the transmission probability $p(r)$ of the links along which the messages are sent such that it can schedule a sufficient number of slots to ensure a reliable transmission.

The outline of the IMPERIA scheduling algorithm is shown in Algorithm 2. It is basically a depth first algorithm operating on the routing tree. Within the procedure *scheduleNode*, a node first aggregates all local messages and adds them to a set of messages *msgs* to be sent. In case it has children, it first collects all the message queues they will send into the *cmsgs* set, then iteratively selects the message queue that is best suited to be aggregated at this moment, and schedules it for transmitting them from the child to the parent.

Note that the message queue is an ordered list of messages, where the last message is still supposed to have room to be aggregated with another one⁹. The *scheduleSlots* allocates slots for communication between the child node and the current node. In case that the message buffer runs full, it adds additional slots to immediately forward all messages to the basestation. Note that the number of slots allocated for the transmission of the message queue is computed using $\lceil count(msg) \cdot E[r] \rceil$, such as to give enough space for retransmissions. An additional consideration here is that the aggregation function used at the sensor nodes must be able to support the kind of aggregation used within this algorithm.

Theoretically it would be possible to schedule multiple sender/receiver pairs during slots within the synchronization

⁹The intermediate messages may also be candidates for aggregation in case they get sent to a node with a small *curmsg*

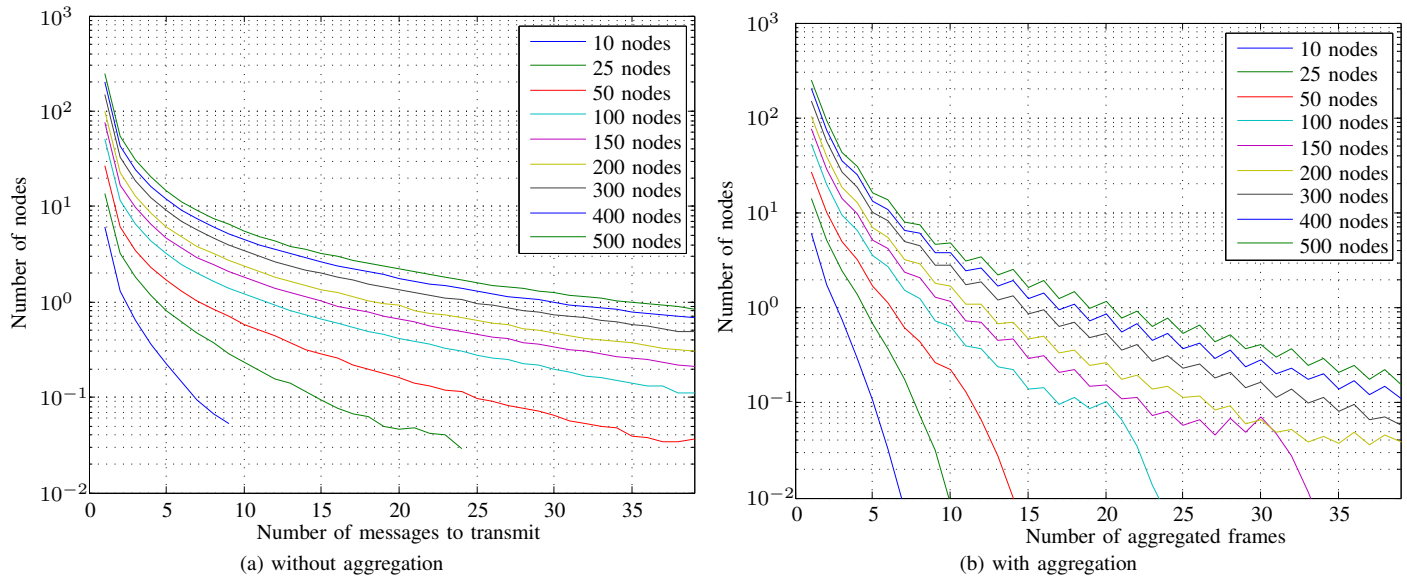


Fig. 6. Number of nodes having to receive or send a number of messages during data collection. Using aggregation allows nodes to transmit fewer messages. The aggregation also increases the probability of a node having an even number of operations.

and collection frames. In large networks, it may be well possible that these pairs do not interfere with each other. It is however difficult to assess which nodes are interfering with each other, as on one hand messages may never actually be received with sufficient quality as to be identified as such, but still disturbing the normal message exchange [18]. Setting up a measurement for interference also requires the different senders to be tightly synchronized to ensure that transmissions actually overlap. Additionally the number of combinations of links that could interfere explodes with increasing network size. Tools such as JamLab [25] exist for this task, but we thus chose for reliability to disallow concurrent transmissions on the same channels. We additionally require clusters to operate on channels that have a distance of at least two channels to minimize also cross-channel interference [26].

Finding schedules of minimal length for the synchronization and collection frames can be reduced to a graph coloring problem, for which solutions exist [27], [28]. Figure 7 compares the network size with the required numbers of slots to transmit its data. For each line, the mean and variance of 1000 randomly generated networks of different size, but equal node density are displayed. The WSNs have been clustered into one (Figure 7a) and four (Figure 7b) clusters with centralized basestations as in Section V.

Figure 7a also includes the broadcast schedule length, which for both cases has the same length. The IMPERIA scheduling algorithm allows only one sender at a time, while Ramanathan [27] and PEDAMACS [28] allow senders that do not interfere at the receivers to send concurrently. The best interference-aware slot assignment algorithm (Ramanathan) reduces the schedule length for four clusters to 43% of the single cluster distribution. The standard deviation however increases by 24%.

From this data it can be concluded that using multiple clusters with centralized basestations reduces the time to

collect data in a similar fashion as allowing simultaneous non-interfering transmissions. Using both mechanisms simultaneously even reduces the duration of the data collection to a value close to the duration of the broadcast frame.

Note that we assume that two sending nodes do not interfere with each other if no link has been found during the topology discovery between the two sender/receiver node sets. This neglects the fact that even if two nodes may not be able to receive messages from each other, they still may interfere with each other. Measuring such interference in a real network is however difficult and costly due to the many possible interference combinations. A safe alternative is always to use the IMPERIA scheduler which does not allow parallel transmissions.

VII. IMPLEMENTATION ASPECTS AND PRACTICAL EXPERIENCES

We have implemented IMPERIA as node network stack for two operating systems: TinyOS 2.1 [29] and IBM's Mote Runner [30] sensor nodes. The IGW, IGC, and IMPERIA broker have been implemented in Java and are executable on embedded computers such as plug computers. Additionally we have visualization clients for desktop computers as well as Android devices. Finally, we developed a network editor which allows experts to edit the automatically generated routing trees and to monitor the health status of the WSN.

In this section we discuss details on the implementations and our experiences with three deployments: (i) our laboratory WSN consisting of up to 24 sensor nodes, (ii) a temperature sensing network for datacenters with up to 108 sensor nodes, and finally (iii) a seismic vibration sensing system which included 38 nodes. Each of the three were built for different purposes and had different requirements on the network stack. In the next sections we discuss how IMPERIA successfully coped with the different requirements.

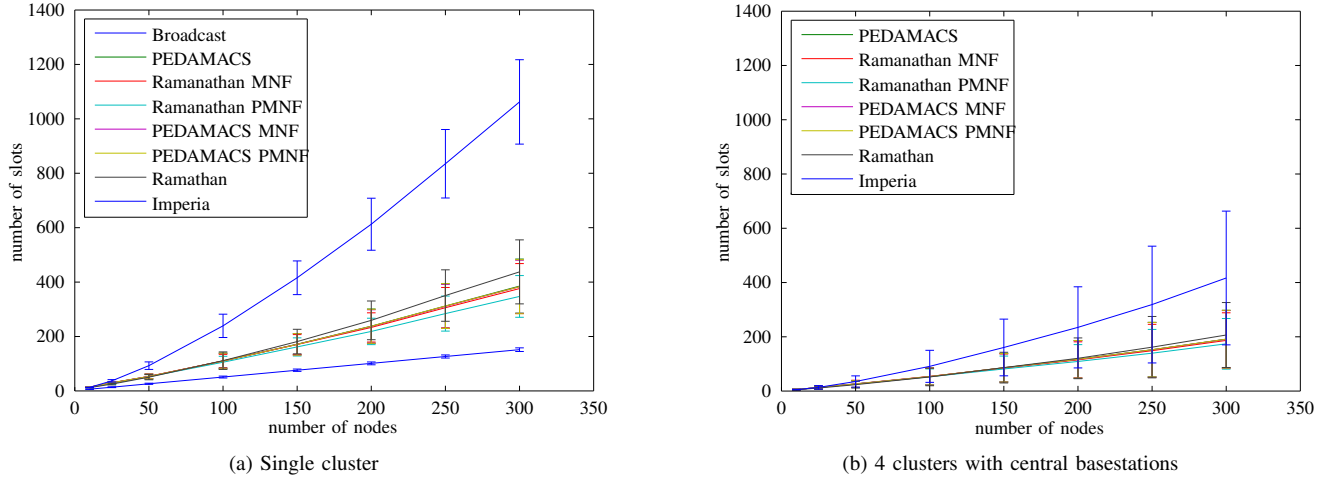


Fig. 7. Number of TDMA slots required for a WSN with a given number of nodes. The number of slots required for the broadcast tree increases much slower than the number required for the collection tree, where a slot contains only one sender and one receiver. The plot shows also the standard deviation σ over 1000 randomly generated networks.

Algorithm 2 The DISTRIBUTE scheduling algorithm

```

scheduleNode(rootNode);
procedure SCHEDULENODE(node)
  msgs = {}
  curmsg = {}
  while localMsgs not empty do ▷ Local data
    nextmsg = selectFittestMsg(curmsg,localMsgs)
    if nextmsg not null then
      curmsg = curmsg ∪ nextmsg
      localMsgs = localMsgs \ nextmsg
    else
      msgs = msgs ∪ curmsg
      curmsg = {}
    end if
  end while
  if hasChildren(node) then
    cmsgs = {}
    for all child of node do
      cmsgs = cmsgs ∪ (child,scheduleNode(child))
    end for
    while cmsgs not empty do ▷ Incoming data
      nextmsgs = selectFittestMsgList(curmsg,cmsgs)
      if nextmsgs not null then
        curmsg = curmsg ∪ nextmsgs.last
        scheduleSlots(nextmsgs.child,nextmsgs.msgs)
        cmsgs = cmsgs \ nextmsgs
      else
        msgs = msgs ∪ curmsg
        curmsg = {}
      end if
    end while
  end if
  msgs = msgs ∪ curmsg
  return msgs
end procedure

```

A. Imperia API

On the sensor nodes, applications access the IMPERIA network stack via a publish/subscribe API that is designed to match the open standard MQTT-S [4]. In general, topic identifiers are negotiated with the broker upon registration of the client, such that subsequent communication does not need to include full topic strings. IMPERIA however makes use of "predefined" topic identifiers, which the IGW maps onto strings before registering with the IMPERIA broker. As a consequence, backend applications can make use of the standard MQTT-S protocol to receive sensor data and to send control messages back to the WSN.

On the sensor node, the IMPERIA API offers six primitives:

- 1) **publish** – Using the publish method, the application can submit its data to the IMPERIA network stack. The publish method takes a topicID and a buffer with the data to be delivered. Optionally, the publish method also takes a timestamp in the sensor node's local time, which it will synchronize and deliver in the time of the gateway.
- 2) **subscribe** – Using the subscribe method, the application can notify the IMPERIA network stack about topics of interest. The bandwidth from the backend to the sensor nodes is limited, but does allow to send configuration data. The application can register a number of topicIDs at this point. An application can only receive data published by a backend application, publications of other sensor nodes are not automatically distributed within the sensor network.
- 3) **publishArrived** – Upon reception of a publication on a subscribed topicID, the application is notified using the publishArrived primitive. With this method, a topicID and the associated data are passed to the application. It is the applications task to discriminate between different actions to be taken according to the topicID.
- 4) **frameStart** – This event notifies the application that a superframe is starting. An application has now a short

timeframe to deliver its data, such that it will still be transferred during the current timeframe.

- 5) **frameEnd** – This event notifies the application that a superframe has just ended, and adds a timestamp of when the next superframe starts. This allows the application to schedule longer processing or sampling procedures, such that the sensor data is ready upon the start of the next superframe.
- 6) **init** – This method is called by the IMPERIA network stack after its initialization. This is a good point for the application to subscribe to the topicIDs.

Client applications may subscribe to the topic `imperia/apps/data/NODEID/TOPICID` to receive sensor data. `NODEID` identifies the sensor node and may include MQTT-S wildcards (e.g. for receiving data from all nodes). `TOPICID` is the topic identifier provided by the application to the API on the sensor node. The client applications may send control data to the sensor nodes by publishing it to the topic `imperia/apps/ctrl/NODEID/TOPICID`. Instead of `NODEID`, "broadcast" could also be used; in this case, the control data will then be distributed to all nodes.

Within the WSN communication, the topic identifiers `TOPICID` are written into the Type field of the IMPERIA message format as specified in Table III ¹⁰.

B. MoteRunner vs. TinyOS

The IMPERIA network stack was implemented in TinyOS and Mote Runner. Both implementations offer the API described above with an operating system specific flavor. While the IMPERIA API is a single interface in TinyOS, it has additional callback interfaces in Mote Runner, such that applications can specify which kinds of events they want to receive.

Applications for Mote Runner are written in Java and compiled into an optimized intermediate language, which is interpreted on the sensor nodes. Hence the code is executed somewhat slower than the platform-specific executable that TinyOS is compiled into. Mote Runner code can however still compete with TinyOS code in synchronization accuracy and follow IMPERIA's timing constraints closely. This is mainly due to its operating system calls allowing to specify times of when the next radio state change should be executed. Hence the IMPERIA code for Mote Runner can schedule the active and sleeping times for the transceiver in advance and does not have to set its own timers as the TinyOS code has to. The Mote Runner operating system takes care of keeping the transceiver in the most power-efficient state. A comparison of the timings and power consumption between the two implementations show that the Mote Runner implementation can even outperform a TinyOS implementation under certain conditions [31].

C. Vibrations sensing system

A first application for IMPERIA was a system to detect harmful seismic vibrations around drilling sites [2]. TinyOS

sensor nodes equipped with accelerometers were placed on critical infrastructure and continuously measured vibrations. These vibrations were locally processed to determine vibrations that are harmful to buildings according to the industrial standard DIN 4150-3. An important requirement for the system was to provide a continuous and complete record of vibration data for insurance reasons. For each second, a set of processed data values had to be stored and reliably transferred to the basestation. In case of an excessive vibration, a one-second window of raw waveform data should additionally be transferred. The latency of this event should be kept low such that workers are able to take appropriate measures upon an alarm.

The requirements on the network stack were to transfer per sensor node 120 bytes of processed vibration data every 10 seconds, and additionally 1'536 bytes of raw waveform data in case of an alarm event. Because of the high data volume to be transferred in case of an alarm, the IMPERIA TDMA superframe (as shown in Figure 2) was extended with an optional frame which allowed a burst transmission from individual sensor nodes through the WSN to the basestation within a single superframe.

In a field test at an industrial plant, a network with 9 vibration sensor nodes and 29 relay nodes were stress-tested for three days. Of the more than 1 million generated data messages, more than 99% reached the basestation over up to 5 hops. Additionally, the waveforms of 387 artificially generated vibration alarms could successfully be transferred.

A complete setup of the vibrations sensing system is currently being used in the Zurich Research Laboratory's showcase room. It includes a vibration sensor, a basestation connected to a plug computer running the IGW and the IMPERIA broker, and an Android tablet computer displaying the sensor health and acceleration waveforms generated by the vibration sensors. The IMPERIA setup has been configured for automatic start and configuration, such that it can be started and run by non-experts. It has now been running reliably for over a year, the biggest issue that remains is the WIFI connection between the tablet computer and the plug computer.

D. Data Center Energy Management

A Mote Runner implementation was deployed with 108 sensor nodes measuring the temperature distribution within a 2'200 m² datacenter during an upgrade of the cooling system [32]. The new cooling system was intended to use free cooling, but had to raise the datacenter room temperature by 3°C. The aim was to reduce the cooling energy consumption from currently 3'600 to 1'500 MWh a year. A continuous monitoring of the temperature within the datacenter during the replacement and the gradual temperature increase should ensure the continuous operation of the datacenter within safe temperature ranges for the equipment.

The deployment ran for 35 days and collected 29 million temperature samples at 10 seconds intervals. At an average sensor node duty cycle of 0.6%, 98% of the generated temperature samples reached the basestation and only 3 nodes had less than 90% end-to-end delivery ratio. The operators of

¹⁰Note here that the value 128 is added by the IMPERIA stack, such that applications have a topic identifier range of 0-127.

the datacenter were given a client application that generated interpolated temperature maps showing the real-time situation within their datacenter.

E. Lifetime measurements

Before having developed IMPERIA, we have used a commercial wireless sensor network with 10 sensor nodes to monitor the temperature within our datacenter. The network has been running for 793 days during which we had to replace multiple times the batteries of the individual sensors. Figure 8a shows the voltage traces as reported by the commercial system's tool. The tool indicated a low battery voltage when the voltage dropped below 2.65 V, upon which the batteries were replaced. This occurred about every 150 to 180 days. Sometimes however we forgot to replace the batteries, and the sensor nodes ran out of battery power. In the longest run, the battery failed after 192 days at 2.46 V. The mean time of the 41 voltage traces shown in Figure 8a to reach a voltage of 2.65 V is 128.9 days with a standard deviation of 7.3 days.

To test the energy consumption of the IMPERIA network stack, we deployed a small network with a basestation and three sensor nodes measuring temperature every 10 seconds. Two sensor nodes were placed outside of the building such that they were exposed to the weather, the third one was placed inside the building as a reference. Due to the fact that IMPERIA sensor nodes draw only little current during sleep mode and have only short periods of a few milliseconds where they draw a few milliamperes, the batteries can recover capacity between usage and hence be used efficiently [33].

Figure 8b shows the voltage traces over the three IMPERIA sensor nodes and the basestation over the course of one year starting in October 2010. All nodes are awake during a 20 ms broadcast frame slot and a 20 ms collection frame slot. Additionally they are active during a listen frame of 40 ms. The nodes sample temperatures every 10 seconds which is also the period of one superframe. The duty cycle of the nodes thus amounts to 0.4%. With a sleep current of $12\mu A$ and an active current of $10.31 mA$, the average current draw is just $53\mu A$.

The nodes 3 and 4 were placed outside and experienced temperatures ranging from $-10.8^\circ C$ in winter to $43.3^\circ C$ in summer. Node 2 on the other hand was left within the building to have a reference with a more stable temperature. The basestation antenna however was placed outside the building, rendering a bad connection to node 2 which as a consequence had a higher duty cycle at times and depleted its battery faster. Clearly visible is also that some voltage is recovered during warmer days. At the time of writing, the IMPERIA test network is running for 330 days. In separate tests, we have experienced that sensor nodes are still operational with voltage readings around 2 V. Under optimal conditions, we expect a runtime of at least 36 months.

F. Laboratory experiments

For testing and debugging purposes we have deployed a network of 24 sensor nodes within our laboratory as shown in Figure 9a. The sensor nodes were placed about 20 cm above the floor at the walls along the hallways. Due to

the low position of the wireless nodes, the wireless links were disturbed by people walking to and from their office or sitting in meeting rooms. During our tests, we experienced more problems during the working hours and found much more stable conditions during the nights or the weekends. This matches observations reported elsewhere in literature and confirmed that we are using a setup that is comparable to other testbeds.

During most tests, node 0 was used as basestation and was usually attached to a plug computer acting as gateway. Occasionally we placed the basestation at different locations for testing. Clustered networks were also tested with a second basestation placed at positions such that the basestations could not directly hear each other and had to be synchronized over multiple hops. Due to the relatively short distances between the sensor nodes, we performed the link probes usually at a reduced sending transmission level to be able to measure a larger difference between the individual links. As expected, this favored the links offering a clear line of sight. The actual data collection however was always run at maximum transmission power.

With IMPERIA, the discovery of the network topology takes on average 18 seconds and an additional 125 seconds was required to probe the quality of the links. The benefit of using a centralized architecture was that we could run the topology discovery once and use the results to test different routing and scheduling algorithms, which we could then deploy afterwards into the network. Also hand-optimized routing trees can be tested for comparison. The collection routing tree created by the IMPERIA algorithms discussed before is shown in Figure 9b.

Figure 9b also shows the measured performance when we ran the network for roughly two work days. The numbers in the grey boxes indicate the percentage of lost messages from the nodes to the basestation. The numbers in the white boxes indicate the percentage of frames lost when transmitting the frames over the link. The mean delivery rate of all sensor nodes was 99.73%.

We have compared the performance of IMPERIA with the Collection Tree Protocol (CTP) implementation for TinyOS [34]. CTP has been used in several projects and shown to perform well on a range of testbeds¹¹. In contrast to IMPERIA it is a decentralized protocol where each node decides on the best next hop for the messages. CTP dynamically adapts to changing link qualities, but also uses ETX to decide on the best hop. During deployment, CTP instantly attempted to find a route to the basestation and in most cases succeeded without problems. The routes changed however in the following minutes until a stable tree was found.

The routing tree that was used by CTP over most of the time is displayed in Figure 9c. The mean delivery ratio over all sensor nodes was 98.45%. CTP had problems with two sensor nodes, 9 and 23, which both kept frequently changing their parents (23 switched between 0 and 10, while 9 switched

¹¹We have used the TestNetwork source available at <http://sing.stanford.edu/gnawali/ctp/>, which we modified to send messages all 5+random·10 seconds. Note that the test application randomized the generation time of the messages, which, in our experience, improves the CTP's performance.

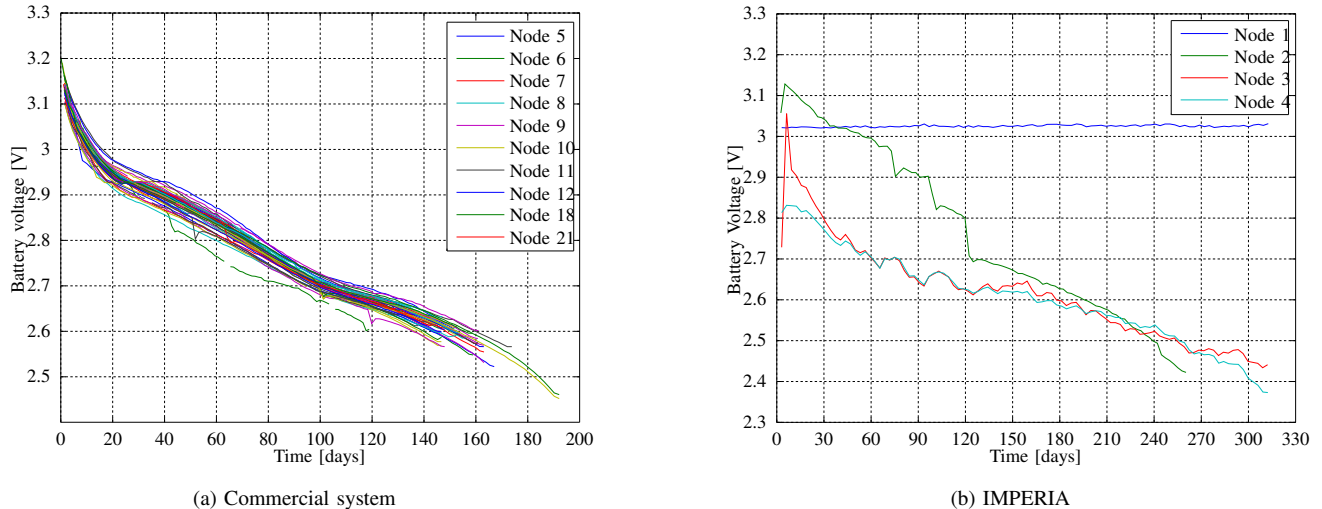


Fig. 8. (a) Commercial system: Battery voltage profile over time for different sensor nodes and multiple runs within a data center. Batteries have usually been replaced after reaching a minimum voltage level. (b) IMPERIA: Battery voltage profile over time for four sensor nodes running IMPERIA during October 2010 to September 2011 outside. The higher variance on the voltage points to larger temperature differences between nights and days.

between 0 and 14). CTP also created a tree that was one level deeper than IMPERIA's; this stems probably from the fact that CTP uses the four-bit link quality to create it [35]. In Figure 9c we have omitted to give the links error rates because the links kept changing and hence could not be compared. We have observed that especially nodes that sometimes have connectivity directly to the basestation kept switching from their parent to the basestation and back.

When comparing our performance to results presented elsewhere in literature we noticed that our laboratory testbed provided relatively favorable conditions. A means for comparison is the Expected Network Delivery (END) metric suggested in [36], which computes the average node-to-basestation delivery rate from all nodes along the globally optimal path and without retransmissions. Our testbed has an END of 0.938 which would be classified as an A+ network. It is important to note however that our network does have a number of unstable links that do not make it easy to find those optimal routes and that our centralized approach provides the tools to ensure that good connectivity can be established and verified at deployment time.

VIII. CONCLUSION

We have demonstrated that a centralized approach to the management of WSNs is an efficient alternative to distributed protocols. Recent advances in the estimation of link qualities allow a better discrimination of unstable links, such that choosing predictable links has become feasible. As a result, networks can run for extended durations with the same configuration and deliver very good performance. Furthermore the benefit of using centrally managed sensor networks comes with the point that they give network operators the possibility to add external knowledge about the environment to their operative decisions, e.g. enforce corrective actions upon the routing algorithms, hence improving the controllability of the networks.

We also discussed efficient algorithms to assess the network topology and obtain link quality information, which is the basis for establishing a network topology. The data obtained can be used by the presented clustering, routing and scheduling algorithms to organize network communication. The gateway software also allows manual corrections or additions. Throughout the report we have discussed details of the protocols and considerations we have taken when designing them. In several deployments and in a direct comparison with an established distributed protocol we have shown that the resulting centrally managed IMPERIA architecture is a viable approach to WSN application design.

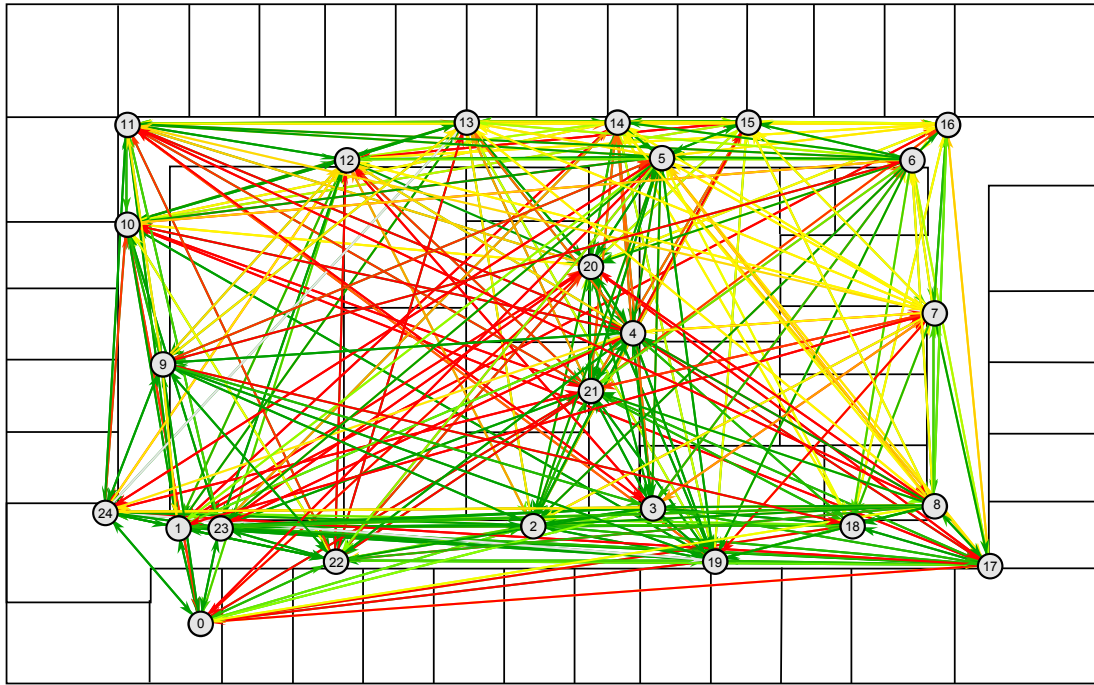
ACKNOWLEDGMENT

The authors thank Wolfgang Schott, Beat Weiss, and Thomas Scherer for their help and discussions during the design and implementation phases of IMPERIA, and Pierre Chevillat for his management support. We also thank Andrea Munari and Eric Bayrakci for their contributions to the routing and scheduling algorithms.

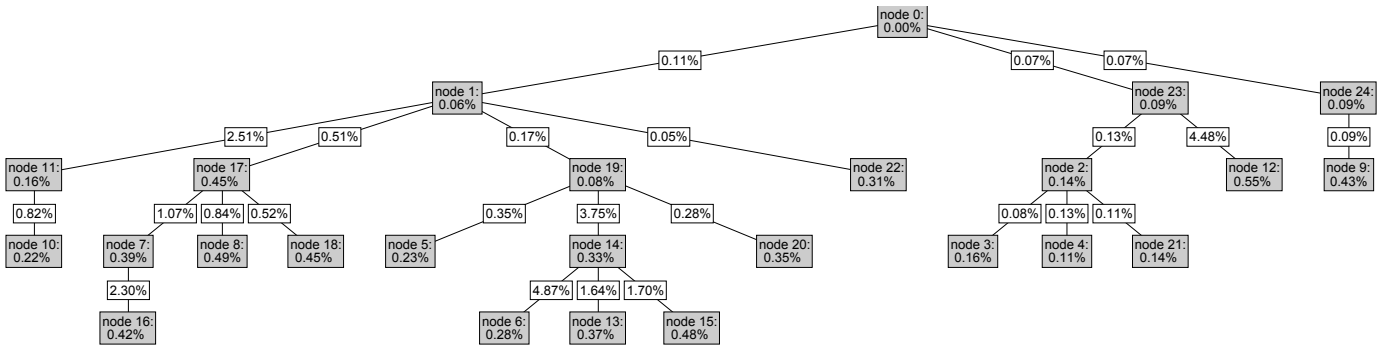
APPENDIX

A. Simulation setup

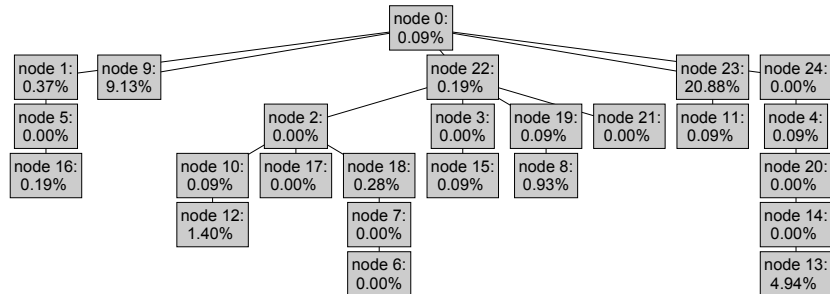
We created wireless sensor network topologies for our simulations by uniformly distributing N network nodes on a square area of $10\sqrt{N}$ units. In case the basestations were placed within the center, they were placed in a horizontally centered line with a distance of 1 unit between the basestations. The communication range between the network nodes was chosen to disc-shaped, i.e. omnidirectional with a radio range of 15 units within which the success rate decreased linearly from 100% to 90% relative to the distance between sender and receiver. We omitted worse links as we believe that links with lower quality would be detected during deployment



(a) Floor plan detailing the sensor locations and link qualities within our laboratory



(b) IMPERIA collection routing tree



(c) CTP collection routing tree

Fig. 9. Comparison of the routing trees created by IMPERIA and CTP. The numbers within the node boxes indicate the percentage of missed end-to-end messages for each protocol. Note that the CTP tree adapted to link quality changes during the runtime, and only the final configuration is shown here.

and omitted by placing sensor nodes at positions with better connectivity or by adding additional relay nodes.

Before simulation, network topologies were tested for being connected, i.e. there existed at least one path, possibly over multiple hops, from each node to any other node. Non-connected topologies were rejected and recreated.

While such simulations are simplistic and may not necessarily appear in reality, through high numbers of repetitions they still generate a wide range of possible network topologies, which correspond to physical deployments. While the physical node locations may not match, the logical link connectivities may do. Most importantly, we have also only used the simulations to get distributions on possible effects over all possible topologies and not to create actual performance results.

B. Topology discovery in dense networks

The amount of neighbor information that a node performing a combined link probe and neighbor discovery has to send back to the basestation may exceed the payload size for dense networks. IMPERIA has the principle that only one message travels through the network at any time, thus sending a burst of data from the node back to the basestation would violate this principle. Instead, a slightly modified version of the topology discovery can be used, which splits up the neighbor discovery and the link probe.

In this modified version, the nodes omit sending the link probe pattern, but instead only broadcast the neighbor discovery messages which include the known neighbors. The nodes then only reply with a list of neighbor addresses, which for IEEE 802.15.4 are the 16-bit short addresses. Depending on the length of the source route back to the basestation, up to about 50 neighbor addresses can be returned to the basestation in this way.

In a second round, the basestation instructs each node individually to broadcast the link probe message pattern. As with the original algorithm, each node receiving such broadcasts collects statistics on the message pattern. After the broadcasts have been completed, the basestation collects the statistics information directly from each node that has been identified as a neighbor in the previous neighbor discovery step.

This algorithm adds additional communication costs to the topology discovery, which depend on the density of the network. In the worst case of a fully connected network, each sensor node needs to be accessed $N + 1$ times, once for the neighbor discovery, and N times for the link probe. Equation 2 then becomes then:

$$T_{discovery} = N((N + 1)E[R] \cdot T_{send} + m_{probe} \cdot T_{send} + m_{discovery} \cdot T_{maxbackoff}) \quad (9)$$

Thus the complexity of network topology discovery has a complexity of $O(N^2)$.

In one of our experiments with a real dense network, we deployed an almost fully connected network of 72 nodes with 4552 links between them. Neighbor discovery needed 22 seconds, the link probe added another 842 seconds, such that it completed after 14:24 minutes. We want to note here that

in this experiment the routing became trivial, because each node could directly transmit to the basestation, but scheduling the transmissions is more challenging. Distributed algorithms using CSMA have severe difficulties in such a dense network, as all the 71 nodes compete for channel access at the same time. Our IMPERIA TDMA protocol on the other hand needed just $(71 + 1)$ slots, which take all together 1.44 s at a slot size of 20 ms.

C. Opportunities for future work

1) Broadcast frame routing respecting clock accuracies:

As the task of the broadcast frame is mainly to synchronize the network, the metric used for building its routing tree may be extended by incorporating information on the clocks of the sender and receivers as proposed in [37]. The clock modules used on different sensor nodes often differ which introduces drift and jitter when comparing their clock domains. The main idea would be to try and match the clock parameters of the synchronization parents and children as closely as possible to improve the synchronization mechanism.

2) *More efficient broadcast frame:* An alternative for a broadcast frame would be to use efficient network-wide flooding to synchronize network nodes, such as Glossy [38]. Glossy requires highly deterministic execution of code such that subsequent transmissions through the wireless medium are highly synchronized (overlap within $< 0.5\mu s$, which leads to a maximum range of links about 150 m). The benefit on the other hand is that it is able to synchronize the whole network within only a few milliseconds. An even faster way would also be to use an external synchronization signal such as an atomic clock reference signal or an AM sender [39]. We chose to use separate slots as an external signal requires additional hardware and the flooding approach makes aggressive use of the wireless communication medium.

3) *Better aggregation:* Currently, aggregation works just by checking whether an incoming set of messages could be added to the most recently added buffer within the message queue. This could be traded off with a more sophisticated algorithm splitting up the set and rearranging the included messages within the queue. This however makes the handling of the timestamps more difficult.

4) *Combined routing and scheduling algorithm:* The routing algorithm presented above does only weakly attempt to load balancing. For a better solution, aggregation and scheduling problems should be considered as well. Aggregation is difficult as an optimal solution may decide to route a message a different way because it can be aggregated somewhere farther down the routing tree, while the number of receive or send slots may be less if using this other path. Each node can additionally reorganize the aggregation of the messages it has in its local buffer before resending them. Additionally, the probabilistic success rate of the channel could also be exploited by rearranging aggregation if a message could not be send.

All these optimization possibilities make an optimal solution arbitrarily complex to compute. Note also that a too optimized version may be heavily vulnerable to message loss as they

may depend on a strict ordering of message transmissions! The aggregation aspect could also be integrated into the routing decision. For example in case of equal EXP, instead of choosing the cluster with the smaller number of nodes, the number of additional slots required for the collection frame could also be considered.

5) *Continuous link probes*: Instead of just monitoring the receptions of the packets at the basestation and periodically obtaining statistics messages from the nodes, a mechanism such as BurstProbe [40] with a special link probe frame could also be used. In environments with low environmental changes (multi-path fading, shadowing effects, or Wi-Fi interference) links are shown to be very stable [41], [18]. In fact, links of good quality (stable/connected) are known to be quite stable [42].

6) *Buffer strategy*: When using unlimited retries on sending a message, finding a good management of the buffers on the sensor nodes becomes a challenge. The WSN characteristics that make it not easy to handle are:

- 1) If the retransmission count is reached within a single time slot, messages may be dropped within a single superframe. It is known that links which are usually very good may temporarily deteriorate, thus not allowing any transmissions for a while. In such a case it is beneficial to just wait for the next superframe for retransmission.
- 2) Furthermore, as long as there are free spots in the buffer messages do not have to be dropped, except if they lose value over time. In such cases unlimited retransmissions would be reasonable as long as they do not block other messages to come through.
- 3) The scheduling should be aware of the buffer problems. If leaf nodes have large buffers and their links get better, they may flood the relay nodes which may have full buffers on their own. Either the relay nodes or the leaf nodes need to drop messages in this case. If the scheduling allows relay nodes to empty their buffers first, this may be avoided.
- 4) When aggregation is used, the problem gets more intricate. Depending on the sequence of incoming messages and the aggregation type used, the number of messages to be sent may vary. The possible combinations of aggregated messages grows with a lower link quality, as the probabilistic effects are increased. Choosing an optimal scheduling that is not conservative, yet ensures not too many buffers overflow is a big challenge.
- 5) Last but not least, buffer handling algorithms should not be too computationally intensive such as to allow maximum message transmission frequency during a slot. Thus only processing times of a few milliseconds can be afforded.

7) *Specialized frames*: The IMPERIA superframe has been designed for flexibility and allows adding frames for specialized functions. For the vibration sensing network, we had added a frame which allows a single node to send a burst of messages to the basestation. That frame could be optionally enabled by the basestation. Another idea presented above was to add a link probing frame. Within such optional frames, only

parts or the whole network may be woken up to perform some function.

Other ideas for specialized frame types include a temporary management mode for parts of the network, such as to transmit configuration information to new nodes, or performing a localized discovery and link probe when the topology changes.

8) *Adaptive superframe duration*: Each time beacon in the broadcast frame includes the relative time until the next superframe will start. This time could be dynamically changed and the network will follow. As links are unreliable, it may be of advantage to have them in a fraction of multiple of the current superframe duration.

9) *Multi-path routing*: To improve reliability, a sender may send its data to multiple relay nodes. This could be implemented in two ways: either multiple receivers are listening during a collection slot, or additional send slots are added for the different parents. The two versions have different drawbacks. Having multiple receivers does not allow acknowledging packet transmissions from both receivers without modification of the physical layer. Having multiple send slots with different parents on the other hand introduces implementation problems in how to handle buffering – should a message still be buffered if just one of the parents has acknowledged it? This may need an introduction of multiple queues, multiplying requirements on scarce memory resources.

In the end, a trade-off between added reliability, and additional complexity and energy cost needs to be considered. Looking at the very good results we already have with single paths, we believe the multi-path option is adding only marginal benefit at a high complexity and energy cost.

REFERENCES

- [1] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung, “MAC Essentials for Wireless Sensor Networks,” *IEEE Communications Surveys & Tutorials*, vol. 12, no. 2, pp. 222–248, 2010.
- [2] B. Weiss, H. L. Truong, W. Schott, A. Munari, C. Lombriser, U. Hunkeler, and P. Chevillat, “A power-efficient wireless sensor network for continuously monitoring seismic vibrations,” in *8th IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2011, pp. 37–45.
- [3] N. Burri, P. V. Rickenbach, and R. Wattenhofer, “Dozer: ultra-low power data gathering in sensor networks,” in *Information Processing In Sensor Networks (IPSN)*, 2007, pp. 450 – 459.
- [4] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “MQTT-S – a publish/subscribe protocol for wireless sensor networks,” in *Proc. 2nd Workshop on Information Assurance for Middleware Communications (IAMCOM)*, 2008, pp. 791–798.
- [5] K. Römer and F. Mattern, “The design space of wireless sensor networks,” *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.
- [6] B. Raman and K. Chebrolu, “Censor Networks : A Critique of Sensor Networks ,” *Computer Communication Review*, vol. 38, no. 3, pp. 75–78, 2008.
- [7] J. P. Lynch and K. J. Loh, “A Summary Review of Wireless Sensors and Sensor Networks for Structural Health Monitoring,” *The Shock and Vibration Digest*, vol. 38, no. 2, pp. 91–128, Mar. 2006.
- [8] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli, “The hitchhiker’s guide to successful wireless sensor network deployments,” in *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys)*, 2008, pp. 43–56.
- [9] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, “Energy conservation in wireless sensor networks: A survey,” *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, May 2009.
- [10] P. Baronti, P. Pillai, V. Chook, S. Chessa, a. Gotta, and Y. Hu, “Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards,” *Computer Communications*, vol. 30, no. 7, pp. 1655–1695, May 2007.

- [11] V. Rajendran, J. Garcia-Luna-Aveces, and K. Obraczka, "Energy-efficient, application-aware medium access for sensor networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, 2005.
- [12] B. Hohlt, L. Doherty, and E. Brewer, "Flexible power scheduling for sensor networks," *Proceedings of the third international symposium on Information processing in sensor networks - IPSN'04*, p. 205, 2004.
- [13] K. Pister and L. Doherty, "TSMP: Time synchronized mesh protocol," *Proceedings of the IASTED International Symposium Distributed Sensor Networks (DSN 2008)*, 2008.
- [14] K. Arisha, M. Youssef, and M. Younis, "Energy-aware TDMA-based MAC for sensor networks," *System-level power optimization for wireless multimedia communication*, pp. 21–40, 2002.
- [15] M. Brownfield, K. Mehrjoo, A. Fayed, and N. Iv, "Wireless sensor network energy-adaptive mac protocol," *Computer Engineering*, pp. 778–782, 2006.
- [16] S. Ergen and P. Varaiya, "PEDAMACS: power efficient and delay aware medium access protocol for sensor networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 7, pp. 920–930, Jul. 2006.
- [17] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter, "Luster: wireless sensor network for environmental research," in *5th International Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.
- [18] L. Mottola, G. P. Picco, M. Ceriotti, . Gun, and A. L. Murphy, "Not all wireless sensor networks are created equal," *ACM Transactions on Sensor Networks*, vol. 7, no. 2, pp. 1–33, Aug. 2010.
- [19] D. Rohm and M. Goyal, "Dynamic Backoff for IEEE 802.15. 4 Beaconless Networks," *IEEE Student Application Paper*, 2009.
- [20] N. Baccour, A. Koubaa, L. Mottola, M. A. Zuniga, H. Youssef, and M. Alves, "Radio Link Quality Estimation in Wireless Sensor Networks: a Survey," accepted for *ACM Transactions on Sensor Networks*, 2011.
- [21] A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Computer Communications*, vol. 30, no. 14–15, pp. 2826–2841, Oct. 2007. [Online]. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0140366407002162>
- [22] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *9th annual international conference on Mobile computing and networking (MobiCom)*. ACM Press, 2003, p. 134.
- [23] K. Srinivasan, "RSSI is under appreciated," in *3rd Workshop on Embedded Networked Sensors (EmNets)*, 2006.
- [24] D. Puccinelli and M. Haenggi, "Reliable data delivery in large-scale low-power sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 4, pp. 1–41, 2010.
- [25] C. Boano, T. Voigt, C. Noda, K. Romer, and M. Zúñiga, "JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *Information Processing in Sensor Networks (IPSN)*, 2011 10th International Conference on, 2011, pp. 175–186.
- [26] Y. Wu, J. a. Stankovic, T. He, and S. Lin, "Realistic and Efficient Multi-Channel Communications in Wireless Sensor Networks," in *27th Conference on Computer Communications (INFOCOMM)*, 2008, pp. 1193–1201.
- [27] S. Ramanathan, "A unified framework and algorithm for channel assignment in wireless networks," *Wireless Networks*, vol. 5, pp. 81–94, March 1999.
- [28] S. C. Ergen and P. Varaiya, "PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks," *IEEE Transactions on Mobile Computing*, vol. 5, pp. 920–930, July 2006.
- [29] P. Levis, D. Gay, V. Handziski, J. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz, "T2: A second generation os for embedded sensor networks," *Telecommunication Networks Group, Technical University, Berlin, Tech. Rep. TKN-05-007*, 2005.
- [30] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich, and I. Romanov, "Mote Runner: A Multi-language Virtual Machine for Small Embedded Devices," in *2009 Third International Conference on Sensor Technologies and Applications*, 2009, pp. 117–125.
- [31] A. Caracas, C. Lombriser, Y.-A. Pignolet, T. Kramp, T. Eirich, R. Adelsberger, and U. Hunkeler, "Energy-efficiency through micro-managing communication and optimizing sleep," in *8th IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2011, pp. 55–63.
- [32] B. Weiss, H. L. Truong, W. Schott, T. Scherrer, C. Lombriser, and P. Chevillat, "Wireless sensor network for continuously monitoring temperatures in data centers," *IBM Research, Tech. Rep. RZ3807*, 2011.
- [33] C.-K. Chau, M. H. Wahab, F. Qin, Y. Wang, and Y. Yang, "Battery recovery aware sensor networks," in *7th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2009, pp. 1–9.
- [34] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 2009, pp. 1–14.
- [35] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four bit wireless link estimation," in *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*, 2007.
- [36] D. Puccinelli, O. Gnawali, S. Yoon, S. Santini, U. Colesanti, S. Giordano, and L. Guibas, "The Impact of Network Topology on Collection Performance," in *8th European Conference on Wireless Sensor Networks (EWSN)*, 2011, pp. 17–32.
- [37] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. B. Srivastava, and P. Dutta, "A Case Against Routing-Integrated Time Synchronization," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010, pp. 267–280.
- [38] F. Ferrari, M. Zimmerling, and L. Thiele, "Efficient network flooding and time synchronization with Glossy," in *10th International Conference on Information Processing in Sensor Networks (IPSN)*, 2011, pp. 73–84.
- [39] A. Rowe, R. Mangharam, and R. Rajkumar, "RT-link: A time-synchronized link protocol for energy-constrained multi-hop wireless networks," in *3rd Annual Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2006, pp. 402–411.
- [40] J. Brown, B. Mccarthy, U. Roedig, T. Voigt, and C. J. Sreenan, "BurstProbe: Debugging Time-Critical Data Delivery in Wireless Sensor Networks," in *8th European Conference on Wireless Sensor Networks (EWSN)*, 2011, pp. 195–210.
- [41] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson, "EM-MAC: A dynamic multichannel energy-efficient mac protocol for wireless sensor networks," in *12th International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2011.
- [42] D. Lal, A. Manjeshwar, H. Falk, E. Uysal-Biyikoglu, and A. Keshavarzian, "Measurement and characterization of link quality metrics in energy constrained wireless sensor networks," in *Global Telecommunications Conference (GLOBECOM)*, 2003, pp. 446–452.