



UbiSec&Sens
FP6-2004-IST-4
Contract no.: 26820



UbiSec&Sens

Deliverable 1.2

Specification and simulation of integrated routing, in-network processing and aggregator node election concepts

Editor:	Evgeny Osipov, LTU Luleå University of Technology
Deliverable nature:	R (Report)
Dissemination level: (Confidentiality)	PU (Public)
Contractual delivery date:	31/12/2007
Actual delivery date:	31/12/2007
Suggested readers:	Experts in the area of wireless sensor networks
Version:	1.0
Total number of pages:	31
Keywords:	Routing and in-network processing, aggregator nodes election, transport protocols, simulations.

Abstract

Deliverable 1.2 presents simulation details for integrated routing, in-network processing and aggregator node election concepts. These concepts are united under the PLUG-IN (PANEL-LUNAR aggregation and routing for in-network processing). The PLUG-IN architecture provides: An integrated solution for universal routing supporting data centric and address centric communications in hierarchical wireless sensor networks with distributed re-election of cluster head nodes. Two variants of reliable transport services developed in the UbiSec&Sens project; and a service for an automatic address configuration.

Disclaimer

This document contains material, which is the copyright of certain UbiSec&Sens consortium parties, and may not be reproduced or copied without permission.

All UbiSec&Sens consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the UbiSec&Sens consortium as a whole, nor a certain party of the UbiSec&Sens consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Impressum

[Full project title] Ubiquitous Sensing and Security in the European Homeland

[Short project title] UbiSec&Sens

[Number and title of work-package] WP1 "Networking"

[Document title] Specification and simulation of integrated routing, in-network processing and aggregator node election concepts

[Editor: Name, company] Evgeny Osipov, LTU Luleå University of Technology

[Work-package leader: Name, company] Evgeny Osipov, LTU Luleå University of Technology

[Estimation of PM spent on the Deliverable] 10 PM

Copyright notice

© 2007 Participants in project UbiSec&Sens

Executive summary

Deliverable 1.2 is the first of two final documents of workpackage WP1 "Networking. It describes simulation details of PANEL-LUNAR aggregation and routing for in-network processing architecture (PLUG-IN). The PLUG-IN architecture provides: An integrated solution for universal routing supporting for both address-centric and data centric communications in hierarchical wireless sensor networks with distributed re-election of cluster head nodes. The D1.2 document overviews the overall PLUG-IN architecture and concentrates on simulations of two integrated chains: PANEL cluster head election protocol, tinyLUNAR routing protocol and GPSR routing strategy; and DTSN, DSDV. It also presents experimental performance studies of NanoTCP and simulations of RSI modules.

PANEL PANEL is a Position-based Aggregator Node Election protocol, meaning that the protocol uses the geographical position information of the sensor nodes to determine the set of aggregator nodes in each epoch during the lifetime of the network.

TinyLUNAR TinyLUNAR is a reactive end-to-end connection oriented routing protocol. It is adapted to the specifics of sensor networks routing scheme originally developed for mobile wireless ad hoc networks. The Lightweight UNDERlay Adhoc Routing (LUNAR) is a layer 2 protocol that utilizes an extended label-switching forwarding technology. TinyLUNAR offers a native support for data-centric and address-centric communications and competitive performance and stability compared to its counterparts, e.g tinyAODV.

DSDV The Destination-Sequenced Distance-Vector Routing (DSDV) implementation developed for UbiSec&Sens is a simplified implementation of the original MANET routing protocol, adapted to the resource restrictions that apply in WSNs. DSDV is a table-driven routing scheme for ad hoc mobile networks based on the classic Bellman-Ford algorithm.

DTSN Distributed Transport for Sensor Networks (DTSN) is a novel reliable transport protocol for convergecast and unicast communications in Wireless Sensor Networks (WSNs). In DTSN, the source completely controls the loss recovery process in order to minimize the overhead associated with control and data packets.

NanoTCP Nano Transmission Control Protocol is a connection oriented end-to-end reliable protocol. It uses a continuous byte stream service between two communicating nodes. In order to provide the service of reliability, NanoTCP has the following features: Multiplexing several applications on top of NanoTCP, connections, acknowledgments and retransmission. Flow control ensured by using the principle of sliding window.

RSI The Robust Self-Initialization (RSI) protocol is a distributed self-stabilizing address assignment protocol for WSNs, which is: i) more energy efficient than previous address assignment protocols (most were developed for MANETS); ii) robust against malicious behavior of a limited amount of nodes; iii) able to join network partitions.

List of authors

Company	Author
BUTE	Gergely Ács
BUTE	Peter Schaffer
INOV	António Grilo
INOV	Carlos Ribeiro
LTU	Evgeny Osipov
RWTH	Christine Jardak

Contents

Executive summary	3
List of authors	4
Abbreviations	7
1 Introduction	8
2 Overview of the PLUG-IN architecture	8
3 Overview of PLUG-IN elements	8
3.1 TinyLUNAR	8
3.1.1 Place of TinyLUNAR and label switching forwarder in the PLUG-IN architecture	9
3.2 PANEL	9
3.2.1 Place of PANEL in the PLUG-IN architecture	9
3.3 GPSR	10
3.3.1 Place of GPSR and geographic forwarding in the PLUG-IN architecture	10
3.4 DSDV	10
3.4.1 Place of DSDV in the PLUG-IN architecture	11
3.5 DTSN	11
3.5.1 Place of DTSN in the PLUG-IN architecture	11
3.6 NanoTCP	11
3.6.1 Place of NanoTCP in the PLUG-IN Architecture	11
3.7 RSI	12
4 Simulation of integrated routing, in-network processing and aggregator node election concepts	12
4.1 Performance of integrated PANEL and TinyLUNAR	12
4.1.1 Simulation setup	13
4.1.2 Radio characteristics	13
4.1.3 MAC layer	14
4.1.4 Node deployment	14
4.1.5 Simulation results	14
4.1.6 Energy consumption of PANEL+tinyLUNAR	15
4.1.7 The Aggregator Node Election Phase	17
4.1.8 The Data Sending Phase	17
4.1.9 The Backup Phase	18
4.1.10 The Query-Response Phase	19
4.2 Performance of integrated DTSN and DSDV	19
4.3 Performance of NanoTCP	22
4.3.1 Parameters definition	24
4.3.2 The NanoTCP performance and evaluation	24
4.4 Performance of RSI	26
4.4.1 Number of messages sent during setup	26
4.4.2 Resilience to badly behaved nodes	28
5 Summary of Deliverable D1.2	29

List of Figures

1	The blueprint of PLUG-IN architecture.	9
2	The integration of PANEL and TinyLUNAR. The wrapper module (PanelToSocketWrapper) communicates with PANEL through the TreeRouting and the Panel interfaces. The former one implements the basic send and initialisation functions, while the latter one provides the basic receive function for Panel that are used by the wrapper to pass messages to PANEL.	13
3	A typical uniformly random sensor network deployment on a square shaped field. The network area is divided into 4 equally sized squares. The connectivity relations between nodes are depicted on the left-hand side, whereas its planarized counterpart is on the right-hand side. The latter one is used by GPSR in case it sticks in local minimas. The network area is 100×100 square meters, the sensor nodes are numbered from 0 to 39, while each base station has id 40, 41, 42, and 43, respectively.	15
4	Average energy consumption for radio transmission (Tx) and reception (Rx)	16
5	The energy consumption of the mcu and radio for different field sizes	16
6	Average completion time of the Aggregator Node Election Phase	17
7	Average delay for different number of delivered packets	18
8	Packet delivery ratios during the Data Sending Phase	18
9	Completion time and packet delivery ratio of the Backup Phase	19
10	Average completion time of Query-Response Phase	20
11	Delivery ratio of the i) query messages and the ii) response messages	20
12	Packet Loss Ratio as a function of the hop distance.	21
13	Throughput as a function of the hop distance.	21
14	Average Delay as a function of the hop distance.	22
15	Average number of RF transmissions per successfully delivered data packet, as a function of the hop distance.	22
16	The multihop-testbed showing the intermediate routers relaying the data and acknowledgment segments between the sender and the receiver.	23
17	The cross-traffic-testbed showing the the additional cross traffic on router R_2	23
18	The distribution in time of the NanoTCP sequence numbers, in a three hop scenario and with various cross traffics.	25
19	The Latency of different NanoTCP segments sizes, in a three hop scenario and with various cross traffics.	25
20	The NanoTCP source data rate and goodput for different sizes of segments and in the presence of various number of hops.	26
21	Impact of the threshold value on the percentage of messages not delivered, with and without power aware rebroadcast delay.	27
22	Energy spent by each node (divided by the energy spent by a single message transmission) for a given coverage.	27
23	Impact of whispering over the percentage of affected nodes, in the presence of a percentage of badly behaved ones.	28
24	Relation between the percentage of failed nodes with the percentage of malicious nodes and network density, with and without whispering.	29

Abbreviations

ACK	Acknowledgement
AW	Acknowledgement window
CLDP	Crossing Link Detection Protocol
DSDV	Destination-Sequenced Distance Vector
DTSN	Distributed Transport for Sensor Networks
EAR	Early Acknowledgement Retransmission
FEC	Forward Error Correction
GG	Gabriel Graph
GPSR	Greedy Perimeter Stateless Routing
LCLR	Lazy Cross Link Detection
LUNAR	Lightweight Underlay Network Ad hoc Routing
MAC	Medium Access Control
MANET	Mobile Ad-hoc Networks
NACK	Negative Acknowledgement
NanoTCP	nano Transmission Control Protocol
PLUG-IN	PANEL-LUNAR aggregation and routing for In-Network processing
RF	Radiofrequency
RNG	Relative Neighborhood Graph
RREQ	Route REQuest
RSI	Robust Self-Initialization
SNR	Signal-to-Noise Ratio
TinyAODV	Tiny Ad hoc On-Demand Distance Vector
TinyDSM	Tiny Distributed Shared Memory
TinyLUNAR	Tiny Lightweigh Underlay Ad hoc Routing
TinyPEDS	Tiny Persistent Encrypted Data Storage
WSN	Wireless Sensor Networks

1 Introduction

Deliverable D1.2 continues to describe the work in workpackage WP1 on creation of a flexible routing and aggregator node election protocol with native support for in-network processing. While the specification of mechanisms designed for this purpose were presented in Deliverable D1.1, this document concentrates on performance aspects of integrated protocols studied through simulations and experiments in testbeds. We present the integrated PLUG-IN architecture (PANEL - tinyLUNAR aggregation and routing for in-network processing). The two approaches that constitute the core of the architecture are PANEL [2], the protocol for distributed aggregator node election; and tinyLUNAR [14], the routing protocol for both data-centric and address-centric communications. The basic configuration of PLUG-IN provides unreliable transport for sensor data in networks with static configuration of node addresses. PLUG-IN(D) provides reliable transport using DTSN transport protocol over DSDV proactive routing. PLUG-IN(N) provides reliable transport through NanoTCP protocol. PLUG-IN(R) provides transport services in networks of interest where addresses are assigned dynamically using RSI address assignment protocol.

The document is organized as follows. Section 2 overviews the PLUG-IN architecture. Section 3 briefly describes the major features of protocols and algorithms included in the PLUG-IN architecture. Simulation details of the architecture follow in Section 4. Section 5 summarizes the deliverable.

2 Overview of the PLUG-IN architecture

The PLUG-IN architecture is shown in Figure 1. The functional blocks shown in the shaded part of the figure represent application and middleware software developed by the UbiSec&Sens project. **TinyPEDS**, the tiny persistent encrypted data storage is a middleware solution provides distributed encrypted data storage within the WSN. Data are encrypted in a homomorphic way in a nested arrangement by applying symmetric as well as asymmetric homomorphic encryption. The sensed data can be aggregated over the time and over the region. Data replica are stored to handle exhausting nodes. They are transmitted to the actual cluster head in the right hand neighborhood. **TinyDSM**, provides means that allow sensor nodes to share their sensor readings in an application dependent way. By that any of these sensor nodes can answer queries for which it has the appropriate data stored. In addition tinyDSM supports an events mechanism, which allows to specify e.g. thresholds and messages that have to be sent in case the threshold is passed. Both components provide high actual in-network processing functionality. Since the implementation details of TinyPEDS and TinyDSM along with supporting security modules are reported in other deliverables, we will not describe further details of these blocks in this document. Note that the implementation details of the supporting Identity component (described in Deliverable D0.2) is reported in another deliverable as well.

The task of the PLUG-IN architecture is to provide low level network support to ensure correct operations of actual in-network processing mechanisms. The architecture fulfills this objective in the following way. We provide two integrated threads within the PLUG-IN solution: TinyPEDS \rightarrow PANEL \rightarrow TinyLUNAR / GPSR \rightarrow Link Layer Label Switching Forwarder; and TinyDSM \rightarrow DTSN \rightarrow DSDV. Where PANEL+TinyLUNAR/GPSR+Link layer label switching forwarding combination in the first chain and DSDV in the second chain independently address the overall objective of the deliverable of creating an integrated flexible routing protocol and an aggregator election mechanisms both supporting in-network processing. Note that while RSI module providing an automatic address configuration does not address the in-network processing functionality, its presence is essential to achieve the deliverable's objective in an autonomic way.

3 Overview of PLUG-IN elements

3.1 TinyLUNAR

TinyLUNAR [14] is the adopted to the specifics of sensor networks connection oriented routing scheme originally developed for mobile wireless ad hoc networks. The Lightweight UNderlay Adhoc Routing (LUNAR) [21] is a layer 2 protocol that utilizes an extended label-switching forwarding technology. The major property of LUNAR is simplicity of implementation in comparison to other protocols developed for MANETs. This is achieved by reducing the route maintenance phase of the protocol to a minimum: In LUNAR all established

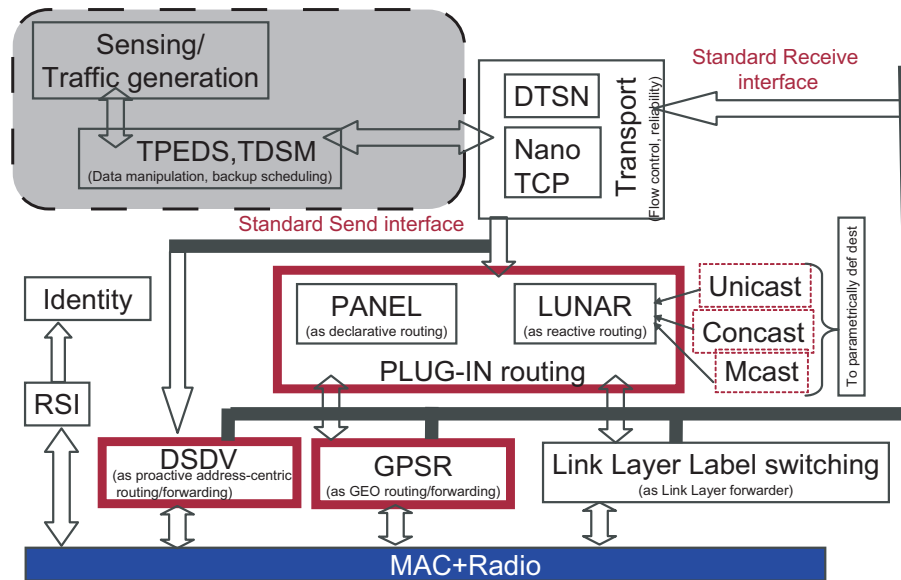


Figure 1: The blueprint of PLUG-IN architecture.

path automatically and periodically expire and rebuild again upon demand from the application. TinyLUNAR inherits the simplicity of its predecessor. In addition it offers a flexible means to the application level programmer to address the destination node parametrically (i.e as a tuple of location and role parameters in case a route is needed to "An aggregator node in cluster X without knowing its actual MAC or other address). The current implementation of tinyLUNAR in TinyOS 2.x for MICA and telos motes offers a competitive performance and stability compared to its counterparts, e.g tinyAODV.

TinyLUNAR is used in conjunction to the Link Layer Label Switching forwarding engine described in [14] as well.

3.1.1 Place of TinyLUNAR and label switching forwarder in the PLUG-IN architecture

TinyLUNAR directly interfaces to upper layers protocols or applications that need a routing functionality in order to reactively establish a multihop path towards parametrically defined destination.

The label switching forwarder is used in conjunction with TinyLUNAR routing logic and other control protocols that use label switching forwarding functionality. In particular, the forwarding component is currently carries control messages of PANEL during the cluster head re-election phase and in-cluster forwarding of data packets.

3.2 PANEL

PANEL [2] is responsible for electing the aggregator node in each cluster of the sensor network. The aggregator nodes are the nodes that collect the measurements of every node in their cluster and aggregate them. Moreover, aggregator nodes store backups of other aggregator nodes. Being an aggregator node means higher energy consumption, therefore a "fair" load balancing scheme is needed which efficiently distributes the load between the nodes in the network. PANEL ensures load balancing by electing the aggregator nodes based on a periodically moving reference point. The node which is the closest one to the reference will be elected as the aggregator of its cluster. All other nodes in the cluster will agree in the ID of the aggregator by receiving the advertisement of the aggregator node and re-broadcasting it.

3.2.1 Place of PANEL in the PLUG-IN architecture

PANEL offers interfaces to upper layer applications that need an aggregator node in the clusters. By request, PANEL initializes the aggregator node election mechanism and performs the election which results in a com-

mon agreement of the ID of aggregator at the end. The return value of PANEL is the ID and position of the aggregator node in a given cluster.

3.3 GPSR

GPSR (Greedy Perimeter Stateless Routing) [8] is a geographic routing protocol proposed for wireless ad hoc and sensor networks that can be used to route data packets between any pair of nodes (i.e., it supports node-to-node communication). GPSR assumes that every sensor node is aware of its own location and the locations of its neighbors.

In GPSR, nodes construct a planar subgraph based on the network topology in a distributive manner. This distributive planarization algorithm can be GG (Gabriel Graph) [7], RNG (Relative Neighborhood Graph) [20], CLDP (Crossing Link Detection Protocol) [9] or LCLR (Lazy Cross Link Removal) [10]. This planar graph will be used to circumvent voids during data forwarding. We implemented GG planarization due to its low communication overhead. Forwarding a packet, each node first tries to greedily select the next-hop towards the base station who is closest to the destination among all neighbors. If all neighbors are farther away from the destination (i.e., there is a local minima) then the forwarding node falls back to perimeter mode by using the planar subgraph to circumvent the local minima.

The clear advantage of geographic routing is that source nodes do not need to flood the entire network to discover routes towards the destination. Hence, it consumes less energy than other flooding-based routing approaches and fits better in lifetime critical applications like the vehicular or agriculture scenario. On the other hand, GPSR requires each node to be aware of its own position and the positions of its neighboring nodes.

3.3.1 Place of GPSR and geographic forwarding in the PLUG-IN architecture

In the PLUG-IN architecture, we use GPSR in two places:

1. The aggregator node election component (PANEL) heavily relies on a geographic forwarding procedure like GPSR. PANEL uses geographic routing to forward data in order to backup in neighboring clusters (a.k.a, inter-cluster routing in PANEL). In particular, at the beginning of each epoch every clusterhead (aggregator node) sends the aggregated data gathered in the previous epoch to a neighboring clusterhead. This is done by geographic forwarding until the packet enters the cluster of the destination aggregator node. Then, the first node reached in the destination cluster can forward the packet further towards the clusterhead using some topology-based routing protocol (a.k.a, intra-cluster routing in PANEL) like TinyLUNAR, or a simple tree routing mechanism.
2. According to the application scenarios, the base station can appear in the network at any location in an undeterministic fashion to collect measurements. In these cases, the base station can send queries to specific clusterheads which then can reply with the aggregated values. The queries of the base station and the replies to these queries are forwarded by geographic routing. In order to preserve valuable memory resources, we reused the code of GPSR to implement the geo-forwarding of these messages.

3.4 DSDV

The Destination-Sequenced Distance-Vector Routing (DSDV) is a table-driven routing scheme for ad hoc mobile networks based on the classic Bellman-Ford algorithm. It was developed by C. Perkins in 1994 [15]. The main contribution of the algorithm was to solve the routing loop problem. Each entry in the routing table contains a sequence number, the sequence numbers are generally even if a link is present else an odd number is used. The numbers are generated by the destination, and the emitter needs to send out the next update with this number. In the original version, routing information was exchanged between nodes by sending full dumps infrequently and smaller incremental updates more frequently. The implementation developed for this project is a simplified implementation of the original MANET routing protocol, adapted to the resource restrictions that apply in WSNs. Instead of full dumps being sent at once, the network nodes periodically broadcast their own routing advertisements and re-broadcast advertisements from other nodes each time they are intercepted.

3.4.1 Place of DSDV in the PLUG-IN architecture

DSDV directly interfaces to upper layers protocols or applications that need a routing functionality in order to proactively establish a multihop path towards one or more univocally addressed destinations.

DSDV is currently integrated with the DTSN transport protocol described below, and is able to support several point-to-point flows simultaneously. The fact that DSDV maintains stable routing paths based on the minimum hop criterium is an advantage for the intermediate node caching mechanism implemented by DTSN. DSDV can also be directly used by applications that do well with best-effort delivery.

3.5 DTSN

The Distributed Transport for Wireless Sensor Networks (DTSN) [13] is a reliable transport protocol developed within this project to provide guaranteed delivery in unicast communications. Although DTSN is based on mechanisms commonly employed in other reliable transport protocols (e.g., ACK/NACK feedback and retransmissions), its efficiency is improved by the following two mechanisms:

- In order to minimize control overhead, the transmission of ACK and NACK packets by the destination is tightly controlled by the sender, which explicitly requests delivery confirmations, piggybacking these requests on data packets whenever possible.
- Since the round-trip-delay is hard to measure in WSNs, the sender maintains a large transmission window, which is divided into smaller acknowledgement windows (AWs) at the end of which a confirmation is requested. Data transmission is only blocked only when the overall transmission window is full. This helps maintaining a continuous data flow.
- Caching of data packets at intermediate nodes serves two purposes: a) to minimize end-to-end retransmissions; b) to increase the probability of delivery of data even if not buffered at the sender (i.e. data that demands partial reliability only, such as precision enhancement data or simply non-critical data).

DTSN also defines a differentiated reliability service where delivery is only guaranteed to a subset of packets in a flow, while the delivery probability of the rest is only maximized based on the caching mechanism eventually coupled with FEC. This advanced service was only specified but not implemented within the project.

3.5.1 Place of DTSN in the PLUG-IN architecture

DTSN offers its upper interfaces to services and applications that require guaranteed data delivery. As an example, within the Homeland Security demonstrator, DTSN will be used to reliably deliver intrusion alerts from the sensors to the sink nodes. Regarding the underlying routing services, the current implementation of DTSN relies on DSDV. However it can be easily adapted to operate with other point-to-point routing protocols.

3.6 NanoTCP

NanoTCP [24] is a simplified version of the reliable byte-stream TCP protocol. Similarly to TCP, each segment contains a source and destination port number to identify the sending and receiving applications. These two values along with the source and destination addresses provided by the lower layers of the stack uniquely identify a connection. The NanoTCP reliability resides in its retransmission capability provided by the sequence and acknowledgment numbers. Congestion control is not considered, as in WSNs losses are very likely due to a bad link quality and this phenomenon invokes erroneous congestion response. The NanoTCP three-way-handshake connection establishment and connection termination work as in TCP. While in TCP the flow of data is controlled by an adjustable window size, in NanoTCP we apply a fixed window size again for simplicity.

3.6.1 Place of NanoTCP in the PLUG-IN Architecture

NanoTCP resides between the application and the network layer. It is used by applications that require a high end-to-end reliability. Moreover, the NanoTCP interfaces with a routing protocol in order to route the segments on a multihop route towards the destination.

NanoTCP segments carry the storing data queries of TinyPEDS from the aggregator node. In order to reach the storage area, segments are routed via two possible routing protocols. First, GPSR is used when routing according to the geographical location of the destination. Second, TinyLUNAR is used when routing according to the destination address.

3.7 RSI

RSI [6] is a Robust Self-Initialization protocol to provide addresses to sensor nodes automatically. The protocol was designed to be:

- Transparent for the sensor node application and for the application developer;
- Energy efficient, in terms of messages and protocol duration i.e. it should use few messages and converge quickly to a global address solution;
- Robust against malicious nodes, i.e. a node should not be able to prevent others from not getting a valid address;
- Detect and solve the network partition joining problem, described in UbiSec&Sens Milestone 1.1.

The module starts to work before the application. It cooperatively chooses an address for every neighbor, and sets the address using the identity module. The module ensures that no two nodes in the neighborhood are equal, however if two previously disjoint networks became visible to each (for instance if a door separating the two networks opens, or more nodes are added to the sensor field) it is possible that a node has two neighbors with the same address. To solve this problem RSI keeps tracking of the messages sent by every neighbor and tries to detect such collisions. If it suspects that an address collision exists it starts the address resolution protocol described in UbiSec&Sens Milestone 1.1.

4 Simulation of integrated routing, in-network processing and aggregator node election concepts

4.1 Performance of integrated PANEL and TinyLUNAR

As we have already described in Section 3.2, PANEL [2] relies on some topology-based routing protocol as an intra-cluster routing protocol. We recall that the task of the intra-routing protocol in PLUG-IN is three-fold:

1. to ensure the delivery of measured data during data gathering. In particular, the routing protocol must ensure that the actual clusterhead receives the measured data sent by the other cluster members.
2. to ensure the delivery of backup data. This means that when a backup message enters a cluster, the routing protocol must deliver this backup message to the actual clusterhead.
3. to ensure the delivery of query messages coming from the base station. Similar to backup messages, the routing protocol must deliver these messages to the respective nodes.

Our aim was to use TinyLUNAR [14] as an intra-cluster routing in the PLUG-IN architecture. In order to use the label-switching mechanism of TinyLUNAR, we first have to create the routing tables at each node. Each routing entry of these tables corresponds to a node that is the current clusterhead or that was a clusterhead in the past. According to the operation of TinyLUNAR [14], an entry contains the id of the next node towards the clusterhead, the id of the clusterhead, and the incoming label of that neighbor. When a new entry is created, the old one is not removed from the table, as we may need it later in case a query released by the base station is destined to a node that was a clusterhead in the past. The construction of the routing entries is as follows. We recall that according to the operation of PANEL [2] each node inside a cluster floods the entire cluster with an announcement message to inform the cluster-members about its own distance from the current reference point. We exploit this flood message to construct the routing entries of each node inside the cluster. In particular, we piggyback the PANEL announcement message to a TinyLUNAR route request (RREQ) message. When a node receives such a message, the routing layer peels the announcement message from the received message,

and passes it up to PANEL, which then updates the destination of the routing entry to the originator of the announcement message. In this way, each node learns who is the current clusterhead, which neighbor is the next node towards that clusterhead, and what is the corresponding incoming label of this neighboring node. When a new clusterhead election procedure is initiated, the routing entries are updated accordingly. Later, this entries can be used to forward data messages, backup messages, and queries to the clusterhead inside a cluster.

This integration process is implemented in a wrapper module which can be considered as an interface between PANEL and TinyLUNAR. The interfaces and their relations are depicted in Figure 2.

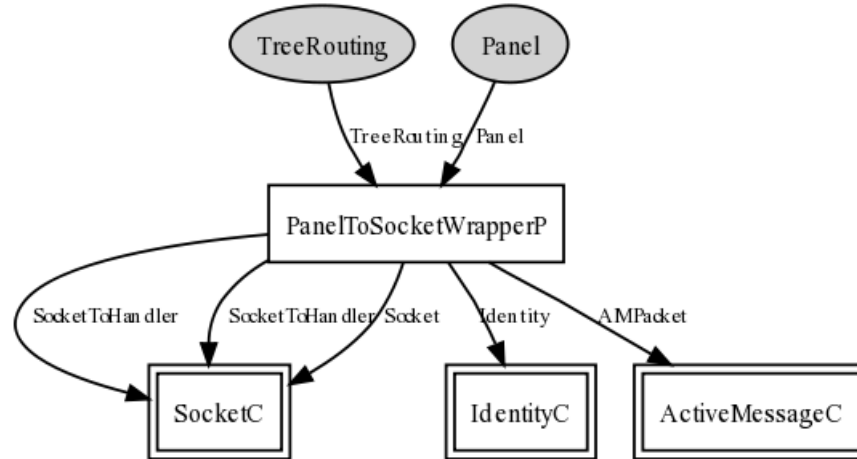


Figure 2: The integration of PANEL and TinyLUNAR. The wrapper module (PanelToSocketWrapper) communicates with PANEL through the TreeRouting and the Panel interfaces. The former one implements the basic send and initialisation functions, while the latter one provides the basic receive function for Panel that are used by the wrapper to pass messages to PANEL.

In this section, we detail the simulation results of PANEL+tinyLUNAR/GPSR. We performed several tests including energy consumption measurement, communication overhead measurement and delay measurement to be able to measure the effectiveness and the performance of the integrated PLUG-IN architecture.

4.1.1 Simulation setup

We use TOSSIM [11] for TinyOS 2 (T2) to measure the performance indices of PLUG-IN. In order to measure the energy consumption, we use the power measurement extension of T2, called PowerTOSSIM [18]. PowerTOSSIM 2 is still experimental and can be downloaded only from the official TinyOS 2 contrib repository [1], however, after minor modifications, we can successfully run simulations. We note that PT2 supports only mica2 [5] which includes the Chipcon CC1000 [4] radio transceiver and the ATmega128L [3] microcontroller.

4.1.2 Radio characteristics

According to the requirements of the UbiSec&Sens agriculture and vehicular scenarios, we assume an outdoor environment with near ground transmissions. Thus, we approximate the environmental parameters according to [19] and [25]. The link gains between nodes are calculated by following the log-normal path-loss model. In this model, the received power at distance d is calculated by formula

$$P_r(d) = P_t - PL(d_0) - 10\eta \log_{10} \left(\frac{d}{d_0} \right) + \mathcal{N}(0, \delta)$$

Here, P_t denotes the transmission (output) power, $PL(d_0)$ is the path-loss at reference distance d_0 , η is the path-loss exponent, and $\mathcal{N}(0, \delta)$ is a Gaussian random variable with mean 0 and variance δ (standard deviation due to multipath effects).

In our simulation model, a packet is delivered from a sender s to receiver r , if the following conditions are satisfied:

<i>Network size</i>	10 × 10	100 × 100	1000 × 1000
<i>Noise level (N)</i>	-105 dBm	-105 dBm	-105 dBm
<i>Output power (P_t)</i>	-20 dBm	-14 dBm	10 dBm
δ	4.2	4.2	4.2
d_0	1 m	1 m	1 m
$PL(d_0)$	36.4 dB	36.4 dB	36.4 dB
Receive sensitivity (S)	-94 dBm	-94 dBm	-94 dBm
$SNR_{threshold}$	10.2 dBm	10.2 dBm	10.2 dBm
η	3	3	3
<i>Node number</i>	40	40	40
<i>Simulation time</i>	7000 s	7000 s	7000 s
<i>Number of simulation runs</i>	20	20	20

Table 1: Simulation parameters for the three different simulation scenarios.

1. $P_r(d) - N < SNR_{threshold}$, where d is the distance between node r and node s , N is the environmental noise, and $SNR_{threshold}$ is the predefined SNR (Signal-to-Noise Ratio) value adjusted by according to the PRR (packet reception rate) – SNR characteristic curve of CC1000 [19]. In all simulation runs, we set $SNR_{threshold}$ to 10.2 dBm which corresponds to 0.9 PRR.
2. $P_r(d) > S$, where S denotes the receive sensitivity of the radio chip,
3. the radio of the receiver is turned on and it is in Rx mode,
4. and the receiver does not receive any other packet simultaneously.

If any of the above conditions does not hold, the packet is dropped.

Our radio environmental parameters are summarized in Table 1.

4.1.3 MAC layer

PowerTossim uses the standard TinyOS 2 CC1000 radio stack, which includes the B-MAC detailed in [16]. B-MAC is a CSMA protocol that employs clear channel assessment (CCA) and packet backoffs for channel arbitration, link layer acknowledgments for reliability, and low power listening (LPL) for low power communication. In our simulation runs, we use the default B-MAC settings (i.e., CCA is enabled, the initial back-off time and the back-off time in case of congestion are chosen uniformly at random).

4.1.4 Node deployment

The simulation setup included sensor networks of 40 nodes where the nodes are grouped in 4 equally sized clusters based on their geographical position. All nodes are pre-configured with their geographical positions. A typical deployment can be seen in Figure 3. As the number of nodes is fixed, the different node densities could be achieved by modifying the size of the field on which the nodes are distributed. The three different fields are square shaped with side lengths of 10, 100, and 1000 meter. For each field size, we performed 20 simulation runs with different uniformly random topologies for each measured quantity. In each case, we ran the simulations for 7000 seconds (i.e., approx. 100 epochs in PANEL).

4.1.5 Simulation results

The simulations we performed aim at a better understanding of the distributed nature and the performance of the integrated PLUG-IN architecture. In order to achieve this, we measured throughout the simulations the following:

- For the whole protocol run
 - energy consumptions of the nodes' radio chip

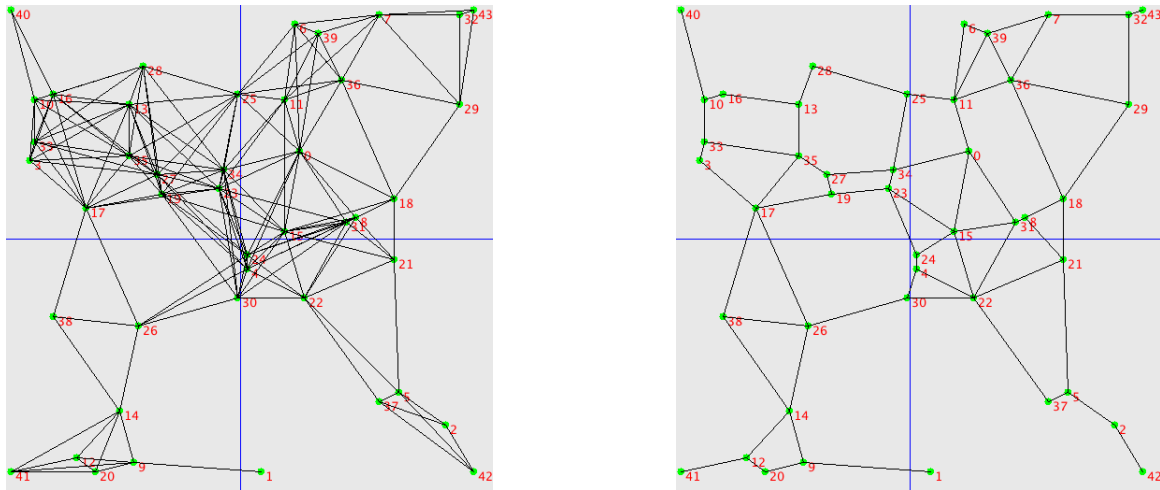


Figure 3: A typical uniformly random sensor network deployment on a square shaped field. The network area is divided into 4 equally sized squares. The connectivity relations between nodes are depicted on the left-hand side, whereas its planarized counterpart is on the right-hand side. The latter one is used by GPSR in case it sticks in local minimas. The network area is 100×100 square meters, the sensor nodes are numbered from 0 to 39, while each base station has id 40, 41, 42, and 43, respectively.

- total energy consumptions of the nodes
- For the Aggregator Node Election phase
 - completion time
- For the Data Sending phase
 - completion time
 - packet delivery ratio
- For the Backup phase
 - completion time
 - delivery ratio
- For the Query-Response phase
 - completion time of the phase
 - query delivery ratio
 - response delivery ratio

In the following, we detail the results of these simulation cases.

4.1.6 Energy consumption of PANEL+tinyLUNAR

The energy consumption of PANEL+tinyLUNAR is measured using PowerTOSSIM 2. The outputs of PowerTOSSIM were processed with Perl scripts which generated Excel tables, and the final figures were generated using Matlab based on the latter tables.

In Figure 4 the energy consumption of the radio communication is shown. The subfigure on the left-hand side shows the energy needed for transmission (Tx) during the 7000 seconds, while the subfigure on the right-hand side shows the energy consumed for reception (Rx). The latter subfigure shows only the energy needed for packet receptions, which means that the energy needs for idle listening and for the overhearing of other nodes' packets are not included in this subfigure. The horizontal axes denote the different field sizes (i.e., different node densities), while the vertical axes correspond to the energy consumption in Joule. The height of the bars

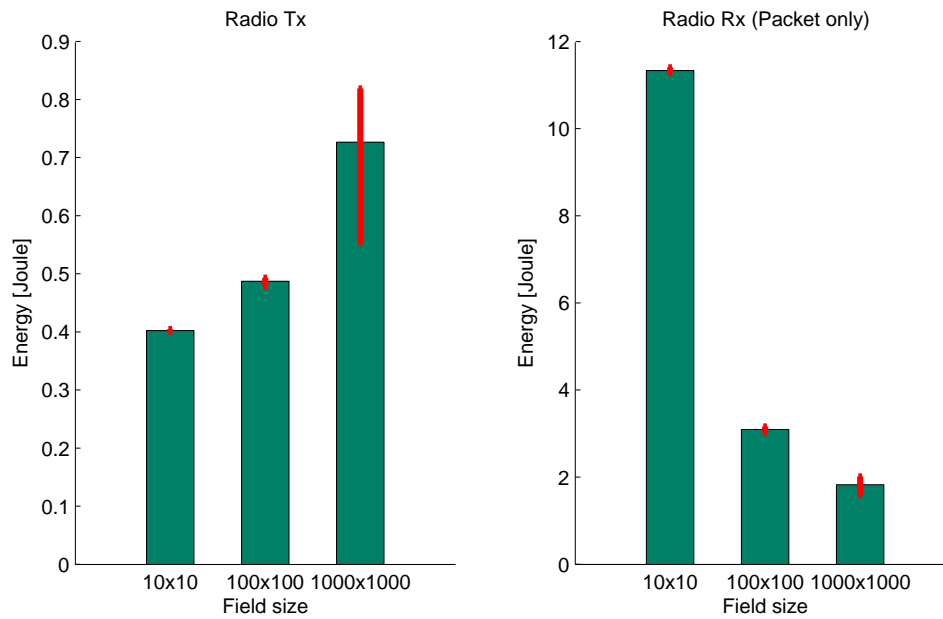


Figure 4: Average energy consumption for radio transmission (Tx) and reception (Rx)

are determined by averaging the energy consumption results for the 20 different random scenarios for each field size (i.e., 10×10, 100×100, and 1000×1000). The whiskers on the top of the bars correspond to the 99% confidence interval of the average.

As one can see, the energy need for packet transmission increases as the size of the area of the network increases. This is natural, since a bigger network has longer routes, which require higher transmission power for the packets to reach the receiver. The right-hand side subfigure is also straightforward to understand: in a sensor network distributed on a bigger field fewer broadcast messages reach the nodes on average, since farther lying nodes may be unable to receive messages from each other. Therefore, the energy needed in a smaller network to receive all the broadcast messages of the other nodes can be saved in this case.

In Figure 5 one can see the total energy consumption of the sensors’ mcu and radio. These results are also averages over the apiece sensors’ results and over the simulation runs for the different field sizes. The horizontal axis of the subfigures corresponds to the different field sizes, while the vertical axis corresponds to the mcu’s and radio’s energy consumption in Joule during the 7000 seconds. The whiskers denote the 99% confidence interval of the averages in total (i.e., mcu+radio). The explanation of Figure 5 is that the energy

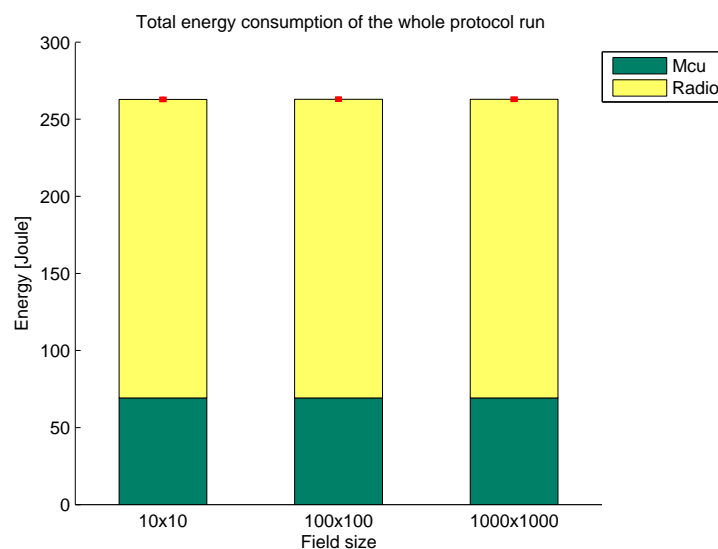


Figure 5: The energy consumption of the mcu and radio for different field sizes

consumption of the sensor's mcu is always very similar independently from the actual topology and its field size. This is natural, since the mcu is only responsible for handling the inner states of the sensor. The similarity in the energy consumption for the different field sizes stems from fact that we did not use energy aware radio techniques, which means that all the nodes' radio was always 'on'. This consumed a huge amount of energy (similarly for each node), but this can be avoided in several ways (e.g., low power listening) in the future.

4.1.7 The Aggregator Node Election Phase

The Aggregator Node Election Phase is the time window when the aggregator nodes in each cluster are elected. This phase starts with the beginning of every new epoch, and ends when all the nodes in a cluster have a common agreement on the id of the aggregator node and each node knows its next hop towards this aggregator node.

In Figure 6 we show the results of the completion time measurements of the Aggregator Node Election Phase. The horizontal axis corresponds to the different field sizes, while the vertical axis corresponds to the completion time in seconds. The bars denote the average completion time of the Aggregator Node Election Phase over the 20 simulation runs and all the epochs in each of these runs. The whiskers correspond to the 99% confidence interval of the averages.

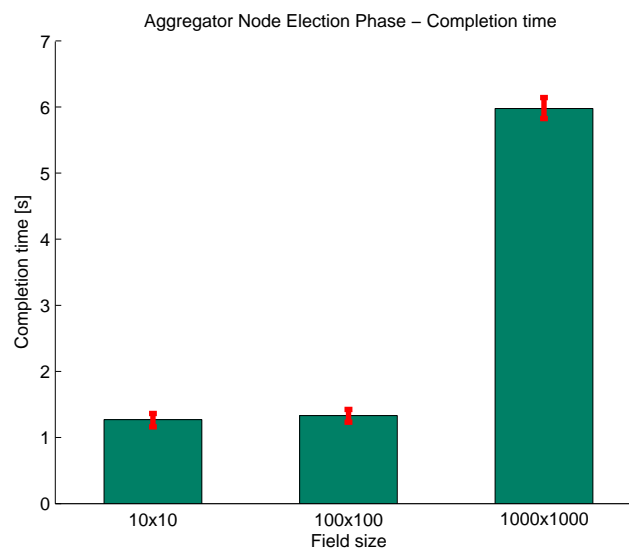


Figure 6: Average completion time of the Aggregator Node Election Phase

As one can see, the completion time for the first two field sizes are below $1.5s$, but for the 1000×1000 field it is about $6s$. This shows that a smaller node density negatively affects the time needed for the common agreement. This is probably due to the "quality" of connectivity: the 10×10 and the 100×100 field sizes imply networks that are either fully connected (each node is connected with every other node, complete graph), or not fully connected but at least connected, respectively. These two cases have similar results, which highly deviate from the case of a network on a field of size $1000 \times 1000 m^2$. This latter case implies a network that can be disconnected: there can be islands composed of nodes (like an archipelago) that do not receive message from others than their members. These archipelagos will wait until the timer of one of the members fires and will finish the aggregator node election process only after that. Archipelagos lying farther may need longer time to complete the aggregator node election phase, which can distort the average to a significant degree.

4.1.8 The Data Sending Phase

The Data Sending Phase is the phase during the protocol run when the cluster member nodes send their measurement data to their own clusterhead. The phase starts five times in each epoch in a regular manner, and ends when all messages from the cluster member nodes in the network are delivered to the clusterhead or get lost.

The delay of this phase is defined as the time needed for all the messages to arrive to the clusterhead or get lost. Obviously, it is important how many messages are delivered when we speak about the delay of a phase that

consists in some message sending and delivering. Therefore, in Figure 7 we show the delay as a function of the number of delivered messages. The three subfigures correspond to the three different field sizes. As expected, the delay grows with the growing number of delivered messages. Moreover, the delay also grows with the field size (except for some very low number of delivered messages).

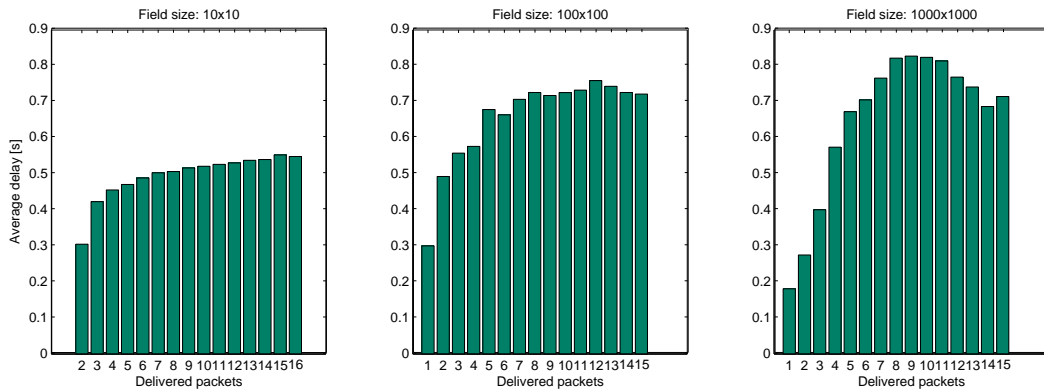


Figure 7: Average delay for different number of delivered packets

Figure 8 is also in agreement with our expectations: a sparse network cannot deliver packets as efficiently as a dense one, since in a sparse network the radio connectivity is weaker and messages can get lost with a higher probability. In Figure 8 one can see the actual numbers of the packet delivery ratio. The horizontal axis corresponds to the different field sizes, while the vertical axis corresponds to the packet delivery ratio. The whiskers on the bars denote the 99% confidence interval.

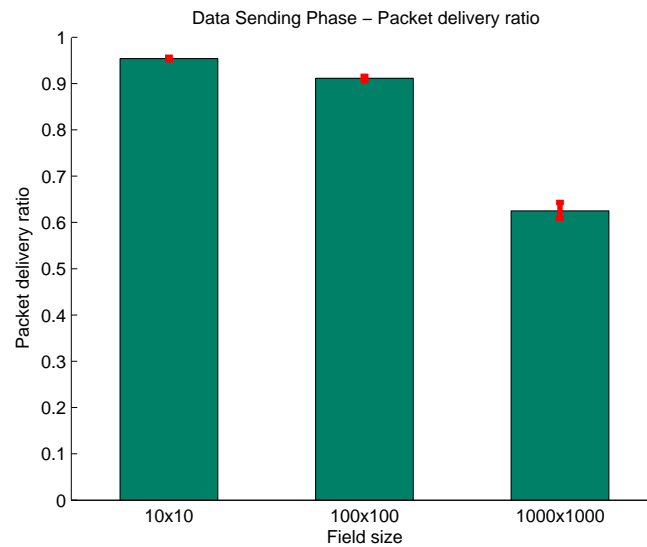


Figure 8: Packet delivery ratios during the Data Sending Phase

4.1.9 The Backup Phase

The Backup Phase is said to be active when a backup is sent and it is travelling towards the backuping node. This phase is initiated at the end of each epoch: It starts with a backup sending, and ends with the reception of the same backup at a different clusterhead (i.e., backuping node).

The two subfigures in Figure 9 show high similarity to Figures 7 and 8. Again, the horizontal axes correspond to the different field sizes. The vertical axis in the left-hand side subfigure corresponds to the completion time of the Backup Phase in seconds, while the same axis in the right-hand side subfigure corresponds to the backup delivery ratio. With a growing field size, the time needed to finish the Backup Phase grows, and in the

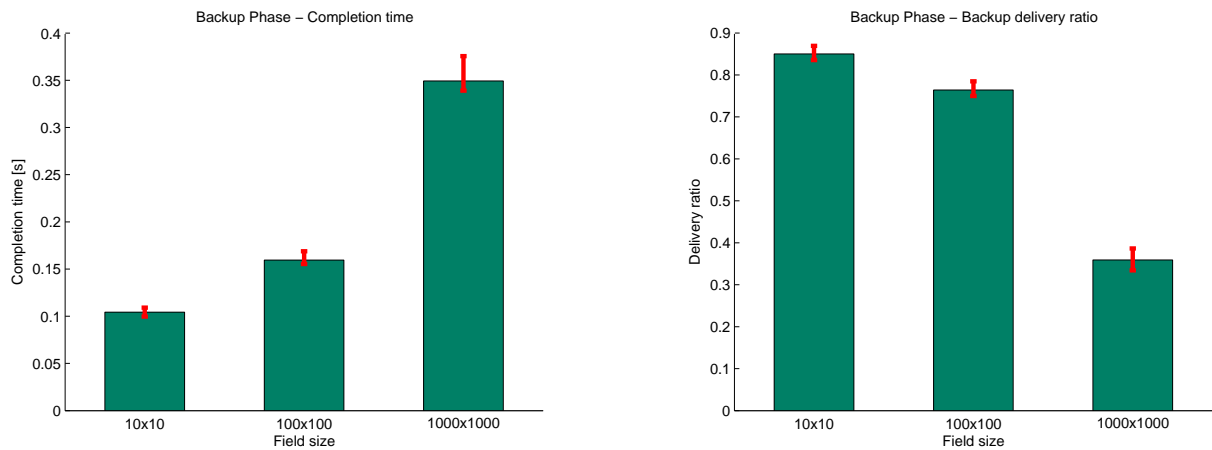


Figure 9: Completion time and packet delivery ratio of the Backup Phase

same time, the packet delivery ratio gets worse. This is natural, but clearly indicates that too sparse networks are not well-supported, since the packet delivery ratio gets even below 40%.

4.1.10 The Query-Response Phase

The Query-Response Phase corresponds to that part in PLUG-IN where the base station queries one of the clusterheads and either that clusterhead or its backuping clusterhead answers the query (or either the query, or the response, or both get lost).

The completion time of the Query-Response phase is defined as the difference in time between the sending of the query and the reception of the response for this query. The completion time is only interpreted for queries that later have a response counterpart received (either with data inside or with the notification that the queried data could not be retrieved). In Figure 10 one can see the completion time of the Query-Response Phase for the 100×100 and the 1000×1000 network. Regrettably, in the case of field size 10×10 no query-response pairs fit the definition above. The reason for this is that we used integer valued coordinates for the nodes which could have disturbed the planarization function of GPSR on this small field. Therefore, in some cases, the planar graph could have been disconnected which led to the failure of response delivery (see also in Figure 11).

The interesting conclusion of Figure 10 is that a sparse network needs fewer time to finish this phase. To explain this one needs to recall that the 1000×1000 networks are sometimes disconnected. The disconnected parts (i.e., archipelagos) are significantly smaller than the whole network, therefore, a query that is broadcast inside an archipelago can quickly find the aggregator node in case the backuping node is located inside that archipelago (since there are only a few nodes to choose from). This shortens the average completion time of the Query-Response Phase for such sparse networks.

The last test case is the query and response delivery ratio test. In Figure 11 one can see these ratios for the different field sizes. Again, the 10×10 networks did not manage to deliver any response messages, that is the reason for the missing bar on the left-hand side. The whiskers on the bars denote the 99% confidence interval. The heights of the bars concur with one's preliminary expectations: the delivery ratio gets worse with the network getting sparser. The previous statement that too sparse networks are not effective is again verified by the fact that the query delivery ratio can go down till 30%. All the same, the response ratio is nearly the same for the 100×100 and the 1000×1000 m^2 field sizes, and it is quite high at approx. 75%.

4.2 Performance of integrated DTSN and DSDV

Simulations of the TinyOS 2.x implementation of DTSN were conducted using the TOSSIM simulation environment. The radiofrequency (RF) parameters of sensor nodes correspond to those of the MICAz motes developed by Crossbow, which support a bitrate of 250 kbps based on the IEEE 802.15.4 standard for the MAC and physical layers. The simulations were made considering a linear topology of evenly spaced sensor nodes, where the transmission power is 0 dBm (maximum power for MICAz nodes), the path loss between each pair

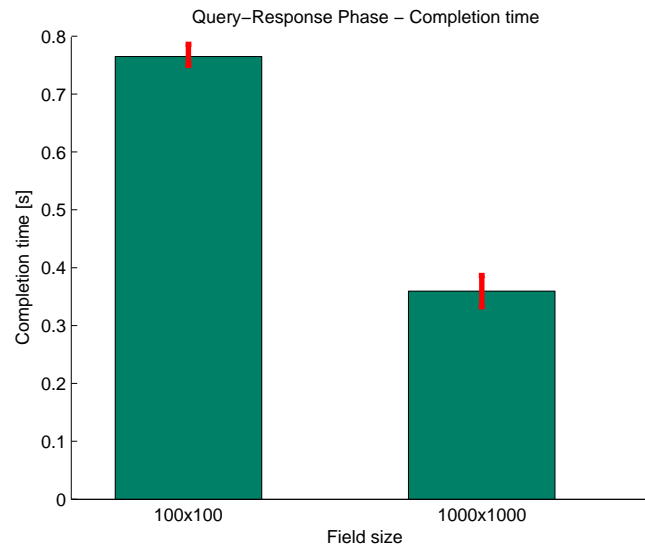


Figure 10: Average completion time of Query-Response Phase

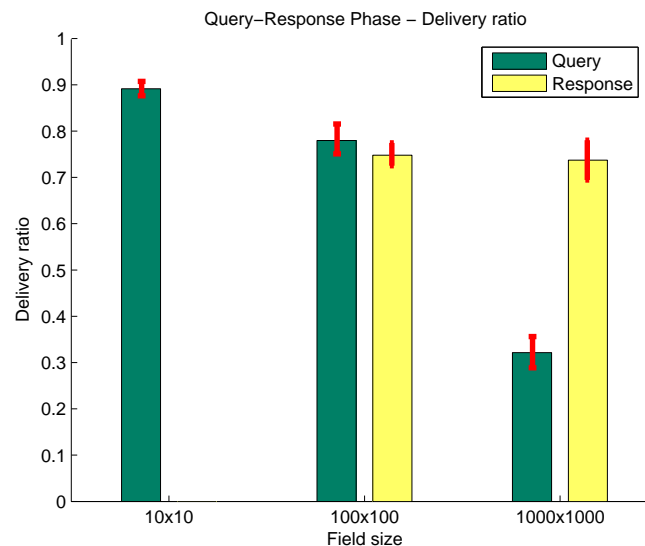


Figure 11: Delivery ratio of the i) query messages and the ii) response messages

of nodes is 70 dB (this corresponds to approximately 10 meters of distance assuming a log-distance path loss model with a distance exponent of 3) and the background noise level is -110 dBm. DSDV was used as the routing protocol underlying DTSN. A continuous traffic pattern was generated at the first node in the line, with the last node as the destination. The data payload size is 2 bytes, adding to the 6 bytes of DTSN overhead and 3 bytes of DSDV overhead. The DTSN transmission window size was configured as 50 packets, with two acknowledgement windows of 25 packets each. The EAR timeout was set as 250 ms. The caching probability p (when used) is set to 1 and the cache size is enough for 50 packets. The results allow a comparison between raw DSDV transmission (without reliable transport on top) and DTSN with and without the intermediate node caching mechanism. Each point in graphs below is an average over ten independent runs, each consisting of the transmission of 1000 packets back-to-back (subject to local interlayer flow control). The achieved packet loss ratio is shown in Figure 12, the throughput dynamics is shown in Figure 13 and the average delay in Figure 14. Finally, the per-packet overhead measured as the total number of radio transmissions per successfully delivered packet received at the destination is depicted Figure 15. All performance characteristics are plotted as functions of the hop distance between source and destination.

The packet loss for DSDV increases quite steadily with the hop distance, justifying the use of a reliable transport protocol like DTSN. However, DSDV still achieves higher throughput and lower delay (the latter, for

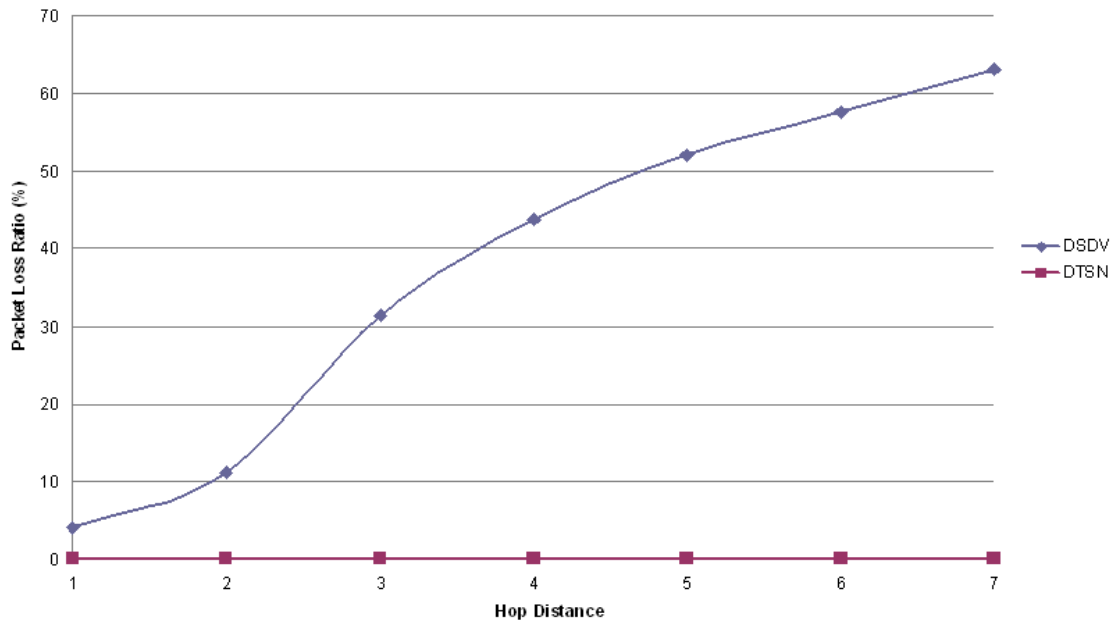


Figure 12: Packet Loss Ratio as a function of the hop distance.

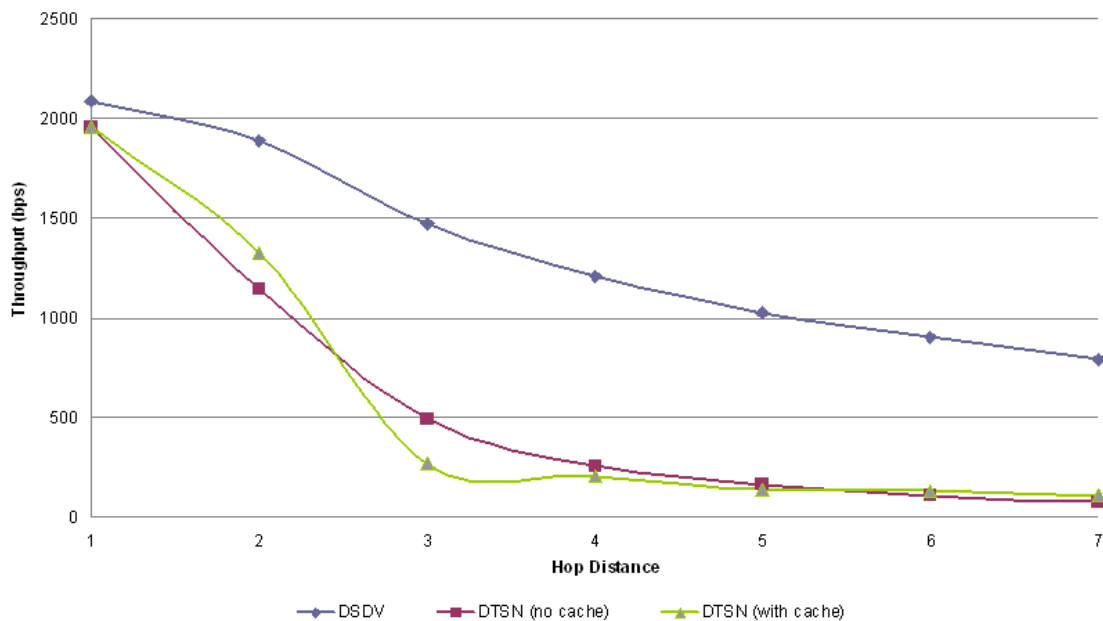


Figure 13: Throughput as a function of the hop distance.

hop distances greater than 3) compared with DTSN, which mimics the difference between TCP and UDP in IP networks. Regarding the overhead, although DSDV clearly presents a lower number of transmissions per packet compared with DTSN without cache, the difference is not very significant when compared with DTSN with the cache mechanism turned on. For a hop distance greater than 3, the cache mechanism starts to pay-off in terms of overhead. In terms of average delay, it only starts to pay-off for a hop distance greater than 5. This surprising result is due to the interference between NACK and data packets retransmitted at intermediate nodes when the cache mechanism is turned on. NACK and data packets are simultaneously forwarded in different directions, interfering with each other and causing a higher NACK loss probability. In fact, the DTSN configuration without cache presented a significantly lower number of EAR timeouts, and the analysis of the

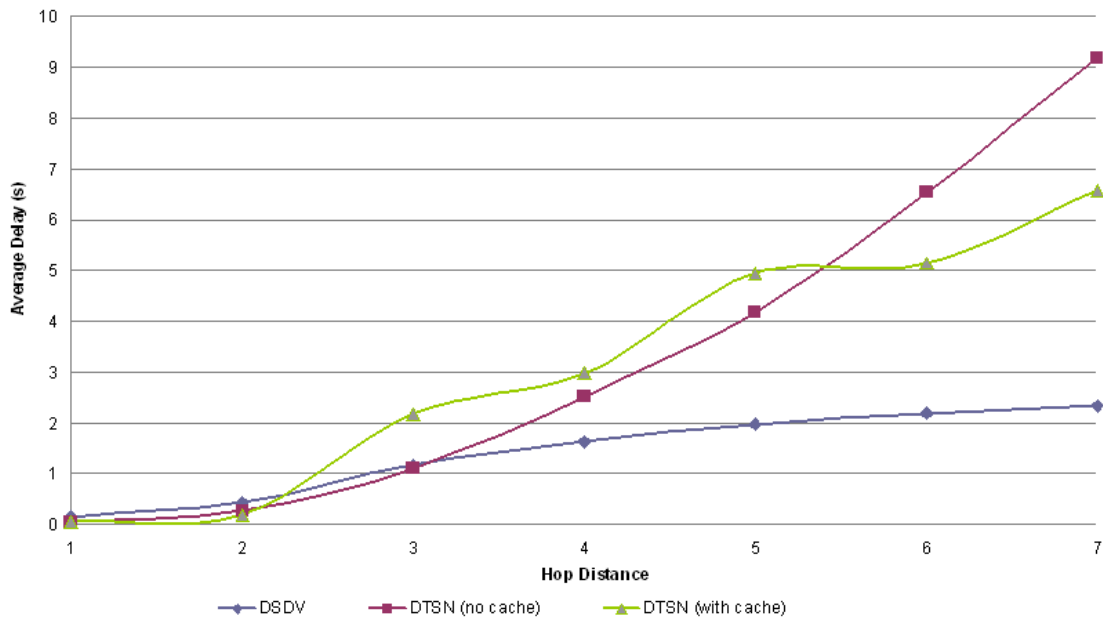


Figure 14: Average Delay as a function of the hop distance.

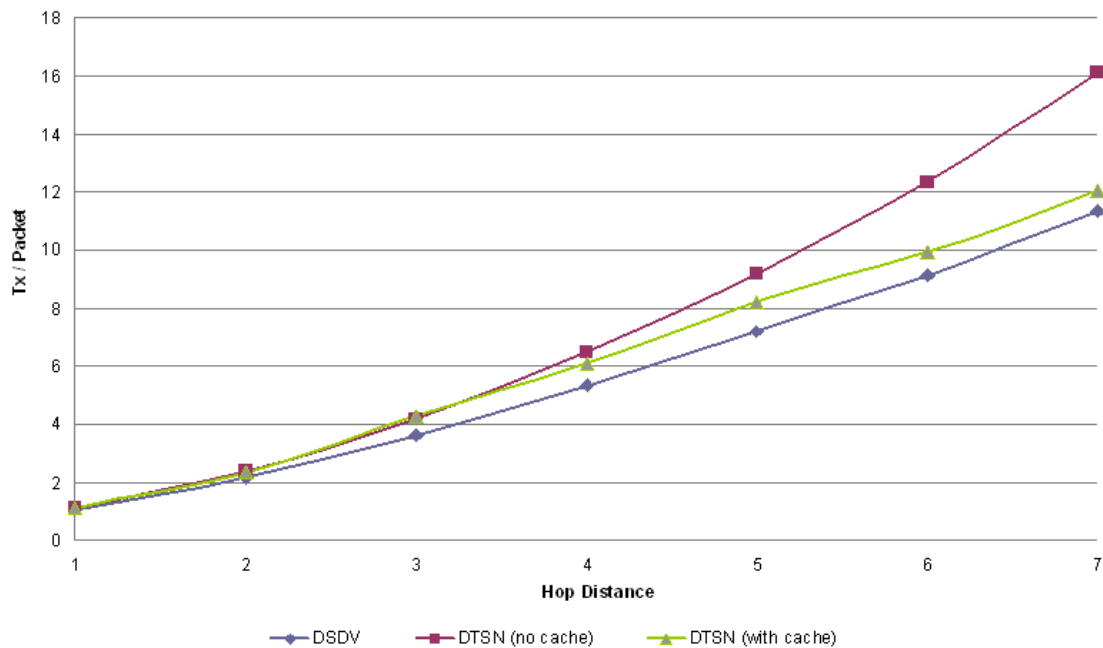


Figure 15: Average number of RF transmissions per successfully delivered data packet, as a function of the hop distance.

simulation traces confirmed this explanation. This problem is currently being investigated so that the caching mechanism is enhanced to avoid this interference.

4.3 Performance of NanoTCP

In this section we report on the performance evaluation of the NanoTCP transport protocol. We first describe the details of the testbeds. Afterward we introduce the parameters to be measured and explain the practical approach used to retrieve the results. We finally comment and analyze the results. NanoTCP is a generic

transport protocol. As such it is independent from the routing protocols running underneath as well as from the user applications. The choice of a transport protocol is strongly based on the needs of the applications. Applications demanding a high level of data reliability, such as code updates, are generally used with reliable transport protocols, e.g NanoTCP. On the other hand, applications relying on the link layer reliability and which are tolerant to data losses can benefit from using a UDP-like protocol. Due to its independence from the routing protocols, we discuss the performance evaluation of the NanoTCP using a simple static routing. Furthermore, we conduct experiments on two testbeds, *multihop-testbed* and *cross-traffic-testbed*.

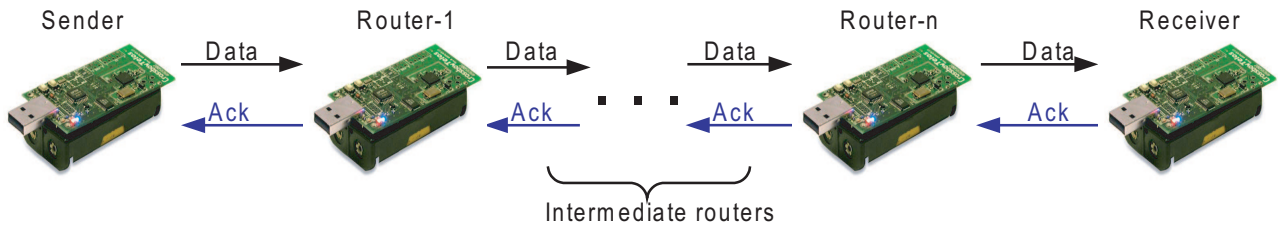


Figure 16: The multihop-testbed showing the intermediate routers relaying the data and acknowledgment segments between the sender and the receiver.

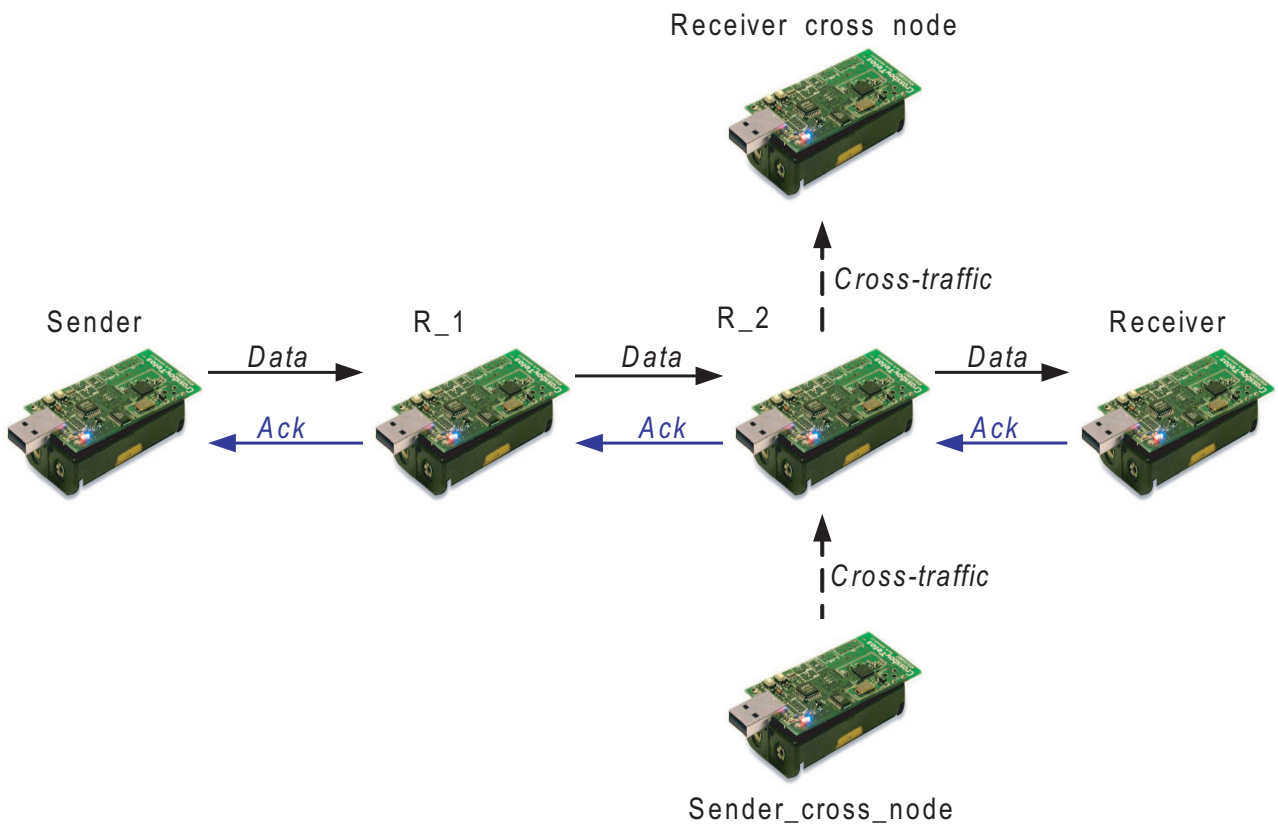


Figure 17: The cross-traffic-testbed showing the the additional cross traffic on router R_2 .

- *multihop-testbed*: We use the *multihop-testbed* shown in Figure 16 in order to evaluate the latency, good-put, delivery ratios and segments retransmission of the NanoTCP as functions of the number of hops. We consider a line topology of motes and to eliminate the losses of MAC packets due to the hidden terminal phenomenon we consider having all the nodes in the range of each other. We install the NanoTCP sender program into the first mote, the receiver program into the last mote and a simple static routing protocol which we have coded on the middle nodes.
- *cross-traffic-testbed*: The *cross-traffic-testbed* shown in Figure 17 is an extension of the previous one

except that we fix the number of hops to three. Having two middle nodes as routers R_1 and R_2 we create packet collisions on R_2 in order to test the behavior of the protocols with segment losses. To accomplish this we insert two new nodes, a node called *sender-cross-node* which sends periodically a 30 byte packet to the other node called *receiver-cross-node* through R_2 .

4.3.1 Parameters definition

In this section we define the measured parameters and explain the practical approach that has been used in order to measure them.

- *Latency*: We define the latency as the needed time for a NanoTCP segment to be successfully delivered to the receiver. The latency is function of various parameters such as the payload length and the number hops. In order to practically measure the latency of a segment, we synchronize the internal clocks of the sender and the receiver. Before being transmitted, the segment is timestamped with the value of the internal clock of the sender. Upon reception, the receiver reads the value of arrival time and calculates the latency as the difference result between both times.
- *retransmissions*: We assume a window retransmission occurring in the case of a lost segment. In order to study this phenomenon, we generate a cross traffic going through one of the routers which relay the NanoTCP segments towards the receiver. The corresponding router, overloaded with relaying the cross-traffic packets, will induce delays in serving the NanoTCP segments and acknowledgments. Such delays will cause timeouts leading to windows retransmissions. We vary the throughput of the cross traffic by increasing the number of packets transmitted during a fixed period of time.
- *throughput and goodput*: The throughput is defined as the number of bits transmitted by the NanoTCP protocol during, T , a fixed interval of time. In our measurements, we take into consideration the bits of the NanoTCP header and the segment payload excluding the B-MAC header. While the throughput is calculated on the transport layer, the goodput is calculated on the application layer. The goodput is the number of bits that the application receives during the same T interval of time. The calculation of both parameters is executed using the similar approach but on different layers. We use counters for calculating the number of bits and in order to acquire the time for a fixed T we use the internal clock time of the telosB.

4.3.2 The NanoTCP performance and evaluation

The NanoTCP is a reliable protocol meaning that it has a 100% delivery rate (barring pathological conditions such as node or route failures or applications trying to send data too fast, leading to buffer exhaustion). The NanoTCP client sends segments in a fixed window size and sets a timer *wait-ack* waiting for the server to acknowledge. If *wait-ack* is fired before the reception of the acknowledgment, the whole window is resent. In our experiment we fix the window size to 2 and we increment the *wait-ack* of 40 ms for each hop. For additional flexibility, in our further implementation we intend to have a dynamic timeout determination for *wait-ack*.

- *Retransmission*: Figure 18 shows the sequence numbers distribution of a single NanoTCP connection. Where 3900 bytes are to be sent to the server, with a 39 byte NanoTCP segment payload. Theoretically 100 data segment have to be transmitted. Practically this connection is subject to different level of losses due to data and acknowledgment segment delays in R_2 . Therefore, in several cases the *wait-ack* is fired before receiving the expected acknowledgment resulting data to be retransmitted. The figure shows the smooth behaviour with the 1.2 Kbps cross traffic rate while the other two have significant variances due to the higher number of retransmissions. As an example on that, we point to the interval of time [3 s,4.2 s] indicated on the figure during which the segment with sequence number 12000 has been several times retransmitted.
- *Latency*: Segment latency is one of the factors highly affected by retransmissions as shown in Figure 19. We have tested two different sizes of segments: *small-size* and *medium-size* segments which correspond 5 byte and 39 byte payload sizes. The former size has obviously a smaller transmission time than the latter one. But the reader notices the drastic increase of both latencies when the cross traffic rate exceeds 5 Kbps.

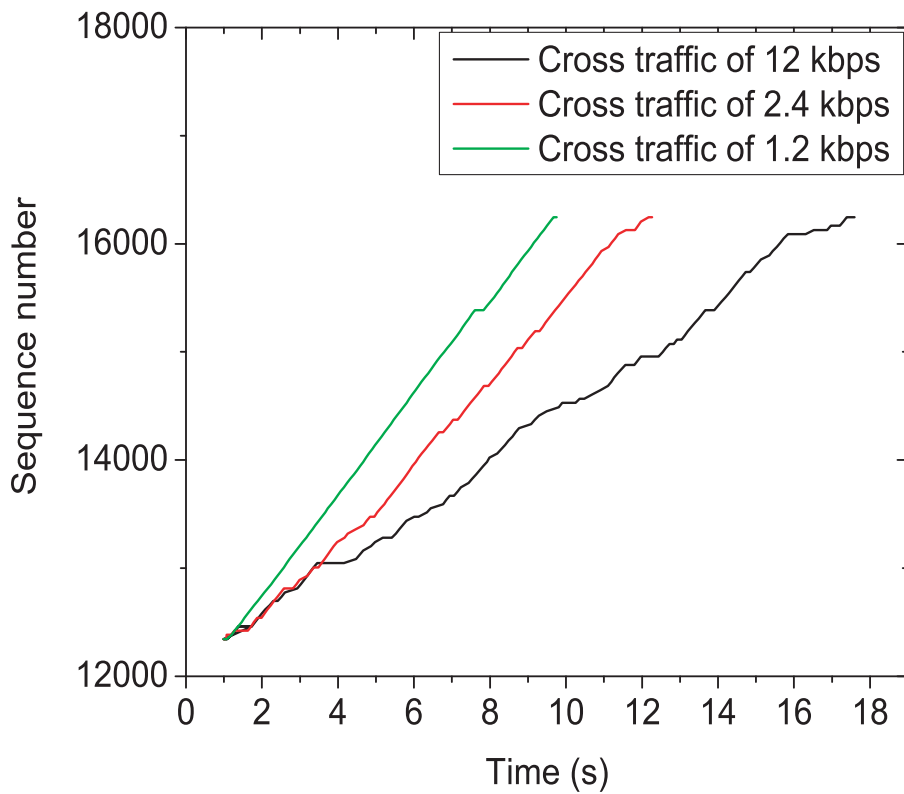


Figure 18: The distribution in time of the NanoTCP sequence numbers, in a three hop scenario and with various cross traffics.

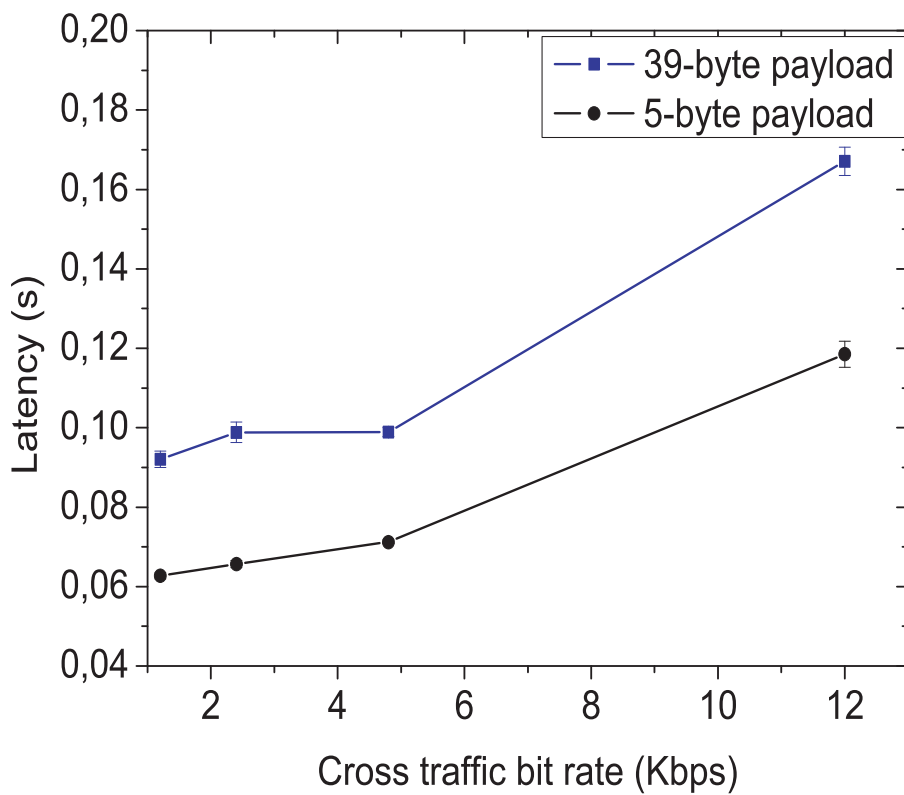


Figure 19: The Latency of different NanoTCP segments sizes, in a three hop scenario and with various cross traffics.

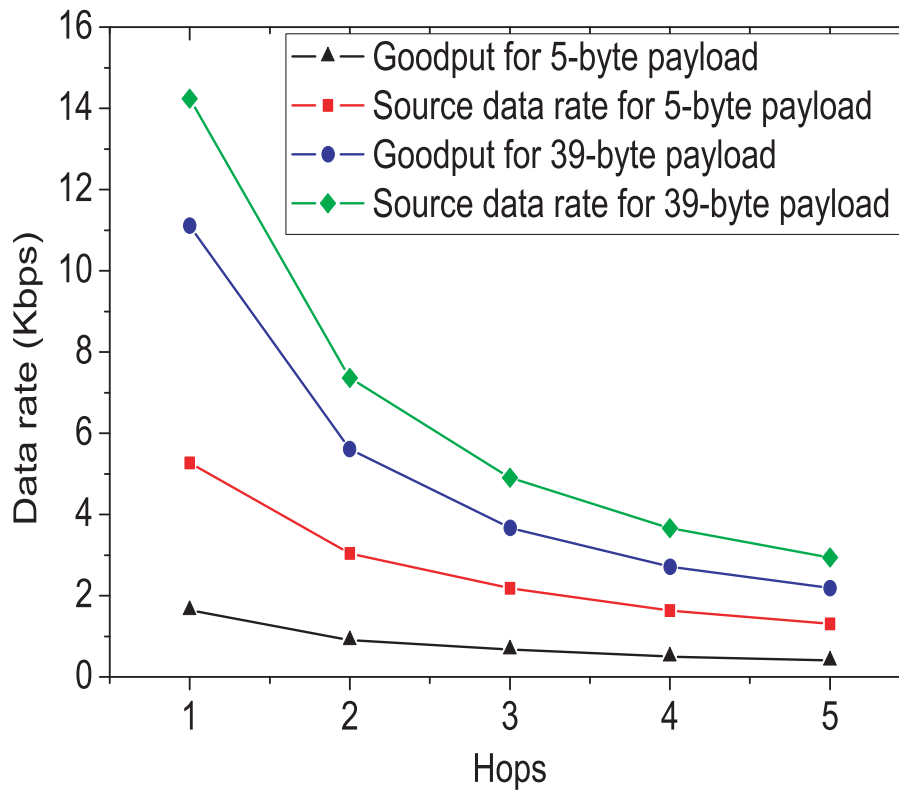


Figure 20: The NanoTCP source data rate and goodput for different sizes of segments and in the presence of various number of hops.

- *Goodput and throughput:* Figure 20 demonstrates the overhead of NanoTCP introduced by its header as well as by the retransmitted segments. It is remarkable that the throughput and the goodput of the *small-size* traffic are less than the corresponding ones in *medium-size* traffic. This is due to the small amount of payload sent in each segment before setting the timer *wait-ack*. Additionally the four curves decrease with the increasing number of hops, due to segment losses and retransmission which increase the latency (explained in Figure 19).

4.4 Performance of RSI

RSI protocol has two distinct phases. The first phase occurs during the initialization of the system and the second phase occurs whenever two disjoint network partitions became visible to each other. The first phase goal is to assign 2-hop unique addresses to sensor nodes, with a minimum number of messages, but ensuring some resilience against badly behaved nodes. The second phase goal is twofold: to detect address collisions with the minimum energy waste and to solve those collisions with minimum impact to routing tables.

The first phase goals can be assessed through simulation, however the second phase goals cannot. The reason for this is the nature of the collision detection mechanism used in RSI. In order to minimize the number of messages, RSI does not periodically send collision detection messages, or send big non-colliding addresses piggyback in other messages as others do [17]. Instead, RSI uses past knowledge of the signal quality of messages from each node to notice any change in the environment, only after that hint RSI sends messages to the node neighbors in order to confirm and solve the collision. We have already some preliminary results on the effectiveness of the approach but those were obtained through a lab prototype.

Therefore the remaining of this section will be devoted to the evaluation of the first phase goals.

4.4.1 Number of messages sent during setup

During the first RSI phase nodes do not have addresses, thus they must contact their neighbors through broadcast messages. Each neighbor should rebroadcast those messages to its own neighbors in order to reach the

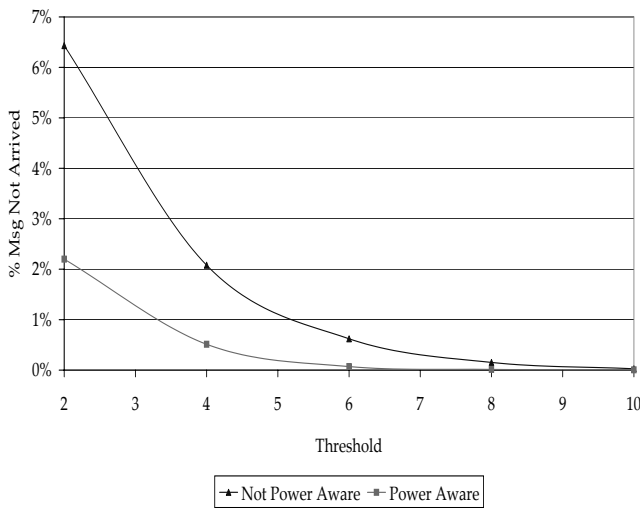


Figure 21: Impact of the threshold value on the percentage of messages not delivered, with and without power aware rebroadcast delay.

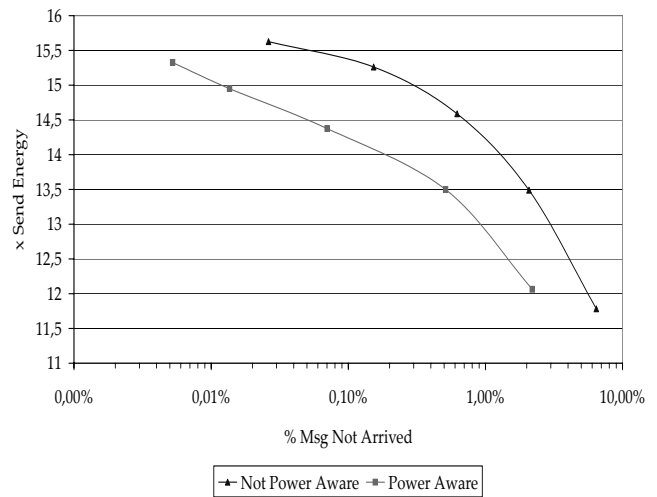


Figure 22: Energy spent by each node (divided by the energy spent by a single message transmission) for a given coverage.

2-hop neighbors of the first one. One of the problems of this solution is that it sends more messages than the ones absolutely needed to reach each of the 2-hop neighbors. There are several solutions for this problem, some more efficient than others, however most are not applicable in these phases, because they require topological information which was not yet obtained.

To reduce the broadcast storm problem we use the counter-based solution proposed in [22] enriched with distance information. In the original counter-based solution some nodes are prevented from rebroadcasting a received message in order to minimize the number of messages sent. Whenever a node receives several replicas of the same message, it concludes that most of its neighbors have already received the message, thus it does not need to send it again. By avoiding sending messages nodes are minimizing the broadcast storm problem and are saving energy but they are increasing the probability of not reaching nodes that they should. In [22], it is shown that, in a homogenous radio network, the uncovered area of a rebroadcast is directly related to the number of copies already received. In the original implementation, nodes rebroadcast after a random delay, provided that in the meantime they have not received enough copies of the same message. In the proposed solution, nodes further away from the source broadcast first, thus increasing the probability that nodes closer to the source are prevented from broadcasting.

There are other methods to minimize broadcast storms with better efficiency ratios, i.e. the ratio between the covered area and the number of broadcasting nodes is better with other methods. However, all these methods require either the knowledge of the topological localization of each node [22] or, at least, each node's neighbors [12].

In the proposed protocol, after receiving a query message, the node checks if that message has been previously received. If the message has been previously received more than a specified number of times, the message is marked as transmitted. Otherwise the message is scheduled for broadcast after a delay directly proportional to the power of the received message. The result is that the retransmission area is divided into concentric rings. The nodes in each of these rings rebroadcast at more or less the same time. Notice that rings are not evenly distributed in space because the reception power varies with the inverse square of the radius, which is more or less consistent with the error in measuring message strength, which is much bigger for low power receptions, i.e. outer rings are wider than inner rings because outer nodes have less accurate positioning than inner nodes.

The first question that arises is the number of copies that need to be received in order to prevent the message to be rebroadcasted. Williams and Cram [23] found that for networks with densities lower than 11 neighbors this threshold must be ≥ 4 to get a maximum coverage, i.e. minimize the number of nodes that never receive the message. However, their scenario is different from our own (we need to cover a 2 hop region while they need to cover the whole network) and they do not use the reception signal strength to schedule rebroadcasts.

The graph in Figure 21 shows the impact on the percentage of uncovered area with the chosen threshold. As expected, the uncovered area decreases with the increase of the threshold. However, it can be seen that the

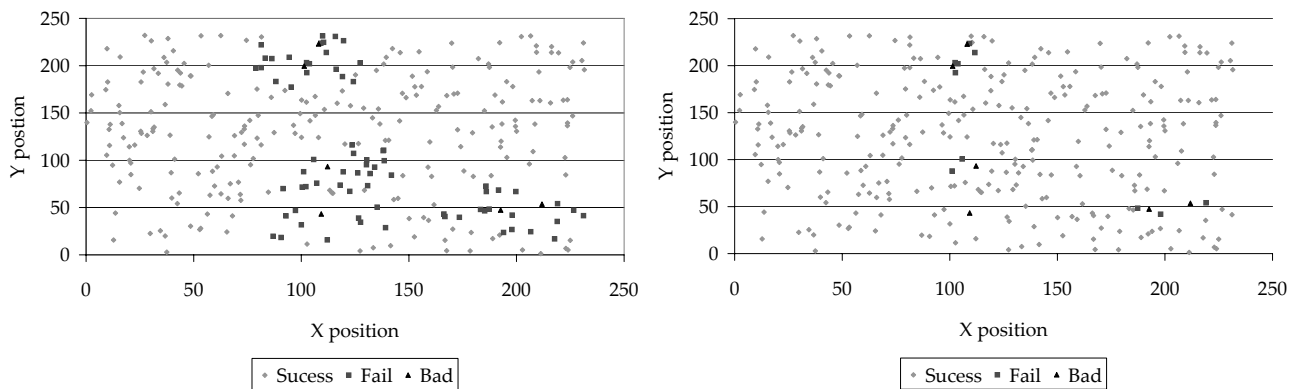


Figure 23: Impact of whispering over the percentage of affected nodes, in the presence of a percentage of badly behaved ones.

threshold required to achieve a significant coverage is much lower with the signal strength information than without it. To get a coverage of 99.5% (i.e. 0.5% of messages not received) we need a threshold of 6 without reception power information and a threshold of 4 with reception power information.

A lower threshold is better because it reduces the number of messages sent thus improving energy consumption and minimizing the broadcast storm problem. In the end, the choice is between energy and coverage. Figure 22 shows the energy spent by each node as a function of the desired coverage. In this graph, we have assumed a simplified energy model in which sending a message consumes one energy unit, the reception of a message consumes 1/10 of a unit and everything else is negligible.

Again, as expected the coverage increases with energy consumption both in the original solution and in the improved one. However, the solution which makes use of reception strength information is able to achieve better coverage with the same energy. In some cases, the uncovered area is 10 times smaller with the same energy consumption.

4.4.2 Resilience to badly behaved nodes

Our protocols starts by choosing an address and asking the node's 2-hop neighborhood if some node already has that address. If one or several misbehaved nodes reply to every query saying that they have already chosen that address, the well behaved nodes may end up with a depleted battery after repeating the query several times. If well behaved nodes do not share individual cryptographic key material with every neighbor, they are not able to distinguish well behaved neighbors from misbehaved ones. In such scenario, the only solution is to progressively reduce the query transmission power until the badly behaved nodes are not able to hear the query. This is similar to whispering to your neighbor to prevent intruders from overhearing.

We have designed RSI with that feature in mind. RSI is not able to completely prevent badly behaved nodes from stopping some well behaved nodes from choosing an ID, but it minimizes the number of affected nodes. Figure 23 shows the effect of a small percentage of malicious nodes (2%) over a field of 300 randomly deployed nodes. Dark triangles represent malicious nodes, light rhombus represent nodes that were able to choose a collision free ID, and dark squares represent nodes that were not able to choose an ID, or if, with the effect of power reduction, became isolated from non-malicious nodes. As expected, the number of nodes which were not able to get an ID using the whispering technique is much smaller than without it. With whispering, the affected nodes are in the direct vicinity of the malicious nodes, while without whispering the affected nodes are spread over the 2-hop neighborhood of the malicious nodes.

The number of affected nodes obviously depends on the number of badly behaved ones, but it is also depends on the network density. The number of failed nodes increases when the number of nodes in the vicinity of malicious ones increases. Figure 24 shows how the percentage of affected nodes increases with the percentage of malicious ones and with the network density. In both cases, the percentage of failed nodes is much lower and increases much slower with whispering than without whispering. In fact, with whispering, the variation of failed nodes with the network density is almost negligible, while without whispering the effect is very noticeable.

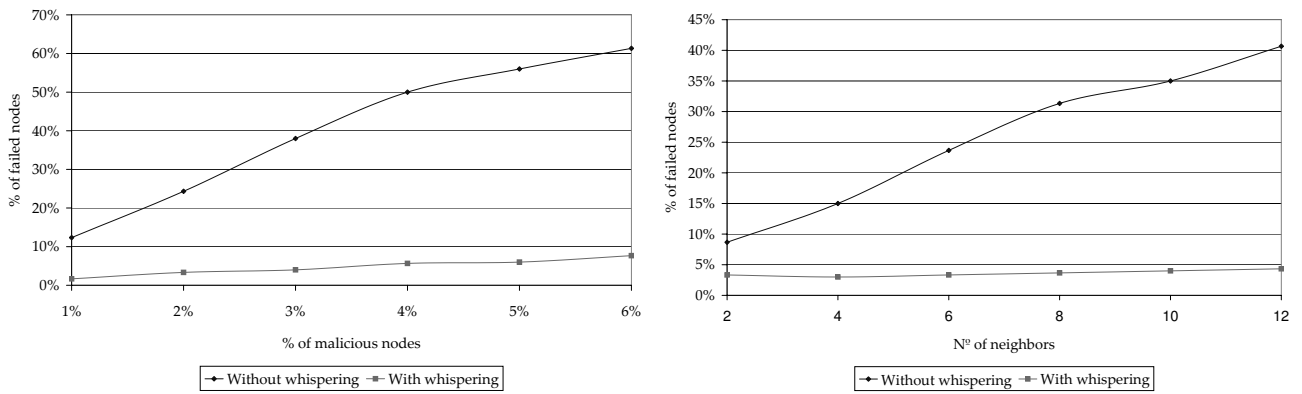


Figure 24: Relation between the percentage of failed nodes with the percentage of malicious nodes and network density, with and without whispering.

5 Summary of Deliverable D1.2

In this document we presented simulation based performance studies of software components in the PLUG-IN architecture. PLUG-IN architecture provides an integrated routing and aggregator node election solution to natively support in-network processing functionality. Two integrated chains constituting the core of the architecture were simulated: PANEL cluster head election protocol, tinyLUNAR routing protocol and GPSR routing strategy; and DTSN, DSDV. We also presented experimental performance studies of NanoTCP and simulations of RSI modules. We showed that the simulated performances of PLUG-IN components confirm their technical characteristics and expectations presented in earlier specifications. The work described in this document gave also essential insights to implementation aspects of the architecture. The implementation details will be reported in the final deliverable D1.3.

References

- [1] TinyOS Alliance. Powertossim for tinycos 2.x.
<http://tinycos.cvs.sourceforge.net/tinycos/tinycos-2.x-contrib/cedt/>.
- [2] L. Buttyan and P. Schaffer. PANEL: position-based aggregator node election in wireless sensor networks. In *Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, 2007.
- [3] ATMEL Corporation. Atmega128l datasheet.
http://www.datasheetcatalog.com/datasheets_pdf/A/T/M/E/ATMEGA128L.shtml.
- [4] Chipcon Corporation. CC1000 datasheet.
http://www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf.
- [5] XBow Corporation. Mica2 datasheet.
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.
- [6] Carlos Nuno da Cruz Ribeiro. Robust sensor self-initialization: Whispering to avoid intruders. In *SECURITYWARE 2007, The International Conference on Emerging Security Information, Systems, and Technologies*, pages 101–107. IEEE Computer Society, October 2007.
- [7] K. Gabriel and R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.
- [8] B. Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless sensor networks. In *Proc. ACM MOBICOM*, Boston, MA, Aug 2000.
- [9] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proceedings of the ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 217–230, 2005.
- [10] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Lazy cross-link removal for geographic routing. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.
- [11] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinycos applications. In *In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.
- [12] Hyojun Lim and Chongkwon Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *MSWIM '00: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 61–68, New York, NY, USA, 2000. ACM Press.
- [13] B. Marchi, A. Grilo, and M. Nunes. Dtsn - distributed transport for sensor networks. In *Proc. IEEE Symposium on Computers and Communications (ISCC'07)*, Aveiro, Portugal, 2007.
- [14] Evgeny Osipov. tinylunar: One-byte multihop communications through hybrid routing in wireless sensor networks. In *New2AN 2007*, September 2007.
- [15] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *SIGCOMM*, pages 234–244, 1994.
- [16] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Boston, MA, November 2004.
- [17] Curt Schurgers, Gautam Kulkarni, and Mani B. Srivastava. Distributed assignment of encoded MAC addresses in sensor networks. In *MobiHoc*, pages 295–298. ACM, 2001.
- [18] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner-Allen, , and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, November 2004.

-
- [19] K. Sohrabi, B. Manriquez, and G. J. Pottie. Near ground wideband channel measurement in 800-1000 mhz. In *Proc. IEEE 49th Vehicular Technology Conference*, Boston, MA, July 1999.
- [20] G. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.
- [21] C. Tschudin, R. Gold, O. Rensfelt, and O. Wiblilng. Lunar: a lightweight underlay network ad-hoc routing protocol and implementation. In *Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04)*, 2004.
- [22] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless Networks*, 8(2-3):153–167, 2002.
- [23] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *MobiHoc*, pages 194–205, 2002.
- [24] Z. Shelby et al. NanoIP: The Zen of Embedded Networking. In *Proc of ICC'03*, pages 1218–1222, Seattle, Washington, USA, May 2003.
- [25] Marco Zuniga and Bhaskar Krishnamachari. An analysis of unreliability and asymmetry in low-power wireless links. *ACM Transactions on Sensor Networks*, 3, June 2007.