A Reconfigurable Approach to the Design of WSN Routing Protocols

APPROVED BY

SUPERVISING COMMITTEE:

Assistant Professor Ioannis Papaefstathiou, Supervisor

Professor Apostolos Dollas

Associate Professor Dionisios Pneumatikatos

A Reconfigurable Approach to the Design of WSN Routing Protocols

by

Georgios-Grigorios Mplemenos, B.Sc.

THESIS

Presented to the Faculty of the Graduate School of

The Technical University of Crete

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

TECHNICAL UNIVERSITY OF CRETE

September 2009

"It is common sense to take a method and try it. If it fails, admit it frankly and try another. But above all try something."

FRANKLIN D. ROOSEVELT

Acknowledgments

This senior thesis was elaborated at the Microprocessor and Hardware Laboratory (MHL) in partial fulfilment of the requirements for the degree of Master of Science from the department of Electronic and Computer Engineering of Technical University of Crete, under the supervision of the assistant professor Ioannis Papaefstathiou.
I feel the need this moment to thank some people who helped me to complete my master thesis, who encouraged me when some obstacles seemed to be insuperable and who finally supported me morally during the last two years.
A big thank is not enough to express my gratitude to my parents and my brother, who encouraged me to this attempt and mainly who supported and continue to support, by all means, me and my choices.
Also, I would like to thank my advisor, assistant professor Ioannis Papaefstathiou for the chance, the excellent working environment and the trust he granted for my research. I hope I didn't let him down with my work - I am still learning things and I am trying to improve my skills all the time!
Furthermore, I am grateful to Associate Professor Dionision Pneymatikatos and Professor Apostolos Dollas who agreed to evaluate this thesis. I would alos like to acknowledge the support of the following people :

- Andreas Brokalakis, research staff at MHL: his experience helped me in many difficult technical issues of this thesis.
- Dimitrios Meidanis, PhD student at MHL, for his great help in learning the tools needed in analog electronics.
- Euripides Sotiriades, PhD student at MHL, for his constant support, during these months, even when everything seemed to be difficult and for the endless moments of snoring during our trips to Finland, Estonia, Belgium and Netherlands - I hope there will be more...!

Also, I would like to thank my friends Konstantinos Papadopoulos, Charis Efremidis, Vasilis Dranos and Stauros Zampelis for all the great student years we

had in Chania, hoping that we will have more of them in the future. Finally, I would like to express my grateful to Zeta for the tolerance she showed the nights I was working on this thesis and for putting up with me. I dedicate this work to her...

A Reconfigurable Approach to the Design of WSN Routing Protocols

Georgios-Grigorios Mplemenos, M.Sc.
Technical University of Crete, 2009

Supervisor: Assistant Professor Ioannis Papaefstathiou

A Wireless Sensor Network (WSN) consists of many sensor nodes deployed in a field, each able to collect environmental information and together able to support multi-hop Ad-hoc routing. WSNs provide an inexpensive and convenient way to monitor physical environments. With their environment-sensing capability, WSNs can enrich human life in applications such as health-care, building monitoring and home security.
The main challenge is to produce low cost and tiny sensor nodes with minimal low power and energy consumption. The large number of sensor nodes involved in WSNs and the need to operate over a long period of time require careful management of the energy resources as most of those nodes are operated by a small battery.
We designed two different WSN platforms: one that consists of a commonly used WNS mote and a CPLD device and one that includes an FPGA and a general purpose CPU. Two different widely used WSN routing protocols were implemented on them and tested in a real WSN environment.
As this thesis demonstrates, we can use low power FPGAs and CPLDs in different kinds of nodes (i.e. sensor nodes or base stations) within a WSN environment, in order to increase the nodes processing power (thus enabling the implementation of more complex functions) and decrease their overall energy consumption.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A Wireless Sensor Network (WSN) consists of many sensor nodes deployed in a field, each able to collect environmental information and together able to support multi hop Ad-hoc routing. WSNs provide an inexpensive and convenient way to monitor physical environments. With their environment-sensing capability, WSNs can enrich human life in applications such as health-care, building monitoring and home security.

The main challenge is to produce low cost and tiny sensor nodes with minimal low power and energy consumption. The large number of sensor nodes involved in WSNs and the need to operate over a long period of time require careful management of the energy resources as most of those nodes are operated by a small battery. Replacing batteries on thousands of WSN nodes may well become infeasible, if not impossible on some applications. Hence, it is well accepted that one of the key challenges in unlocking the potential of such sensor networks is to decrease their energy consumption so as to maximize their post-deployment active sensing lifetime.

Moving to a different field, Field-Programmable Gate Arrays (FPGAs) are semiconductor devices that can be configured by the customer or designer after manufacturing and they perform certain processing intensive tasks more efficiently than the general-purpose CPUs. FPGAs contain programmable logic components called logic blocks, and an hierarchy of reconfigurable interconnections that allow the blocks to be wired together. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complex memory structures.

Moreover, FPGA manufacturers have constructed small Complex Programmable Logic Devices (CPLDs) which are integrated circuits with architectural features of both Programmable Logic Arrays (PLAs) and FPGAs. The main characteristics of the CPLDs are: a) very low energy consumption, b) relatively high bandwidth when executing certain data manipulation tasks and c) low cost (less than $10).

On the other hand, the main disadvantage of those devices is their small number of resources allowing them to execute only small, yet very CPU intensive, tasks. As this thesis demonstrates, we can use FPGAs and CPLDs in different kinds of nodes (i.e. sensor nodes or base stations) within a WSN environment, in order to increase the nodes processing power (thus enabling the implementation of more complex functions) and decrease their overall energy consumption.
To the best of the author's knowledge, this is one of the first attempts to utilize reconfigurable devices in the actual data and networking processing of the WSN nodes and as our real world measurements clearly demonstrate, this approach can be very promising in reducing the overall energy cosumption of a WSN infrastructure at the cost of a few inexpensive CPLD/FPGA devices.

1.2 Scientific Contribution

The contribution of this thesis is the following:

- Design and implementation of 2 newly introduced reconfigurable WSN platforms. The first one includes a common WSN mote and a CPLD device while the second includes a state-of-the-art FPGA device and a general purpose, state-of-the-art CPU.
- Hardware Architecture designs of the commonly-used GPSR WSN protocol (r-GPSR), and implementations on the new WSN platforms.
- Design and implementation of the XMesh's cost function on the CPLD-based WSN platform
- Study and architectural design of a Trust Model for the GPSR protocol. This model adds security and trustworthiness on it.
- Evaluation of the r-GPSR and XMesh protocols on the new platforms based on execution, energy and power attributes. In general, this is one of the first works that studies and proposes a way that reconfigurable devices (CPLDs and FPGAs) can be used in the design of WSN nodes and provides the basis, through the reconfigurable routing protocols, to develop applications that utilize all the advantages of FPGAs.

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows:

- **Chapter 2** presents the theoretical background of WSN routing protocols and a short classification of them based on their attributes and characteristics. Furthermore, this chapter illustrates the state-of-the-art trust models that are used to secure WSN routing protocols. Finally, it summarizes the related work that has been done so far, regarding WSN platforms that utilize in some way reconfigurable devices.

- **Chapter 3** describes in detail the GPSR routing protocol and especially its attributes, its characteristics and the algorithms that implements in order to route packets inside a WSN. Also, in this chapter are presented the software implementations of this protocol for different platforms. A newly introduced trust model is illustrated together with its implementation details for the GPSR protocol. Finally, another multi-hop routing protocol (XMesh) is briefly described.

- **Chapter 4** points out the implementation details of CPLD-based WSN Nodes.

- **Chapter 5** presents the implementation details for the newly introduced FPGA-based WSN node for GPSR protocol.

- **Chapter 6** presents the implementation details for the newly introduced FPGA-based WSN BaseStation for GPSR protocol.

- **Chapter 7** describes the different verification processes that were followed in order to fully test out new platforms. Some monitoring tools that were used during the development processes are also presented

- **Chapter 8**, finally, provides performance results of these new platforms derived from experimental, real world experiments; conclusions and future work on reconfigurable WSN platforms are also presented.

Chapter 2

Theoretical Background and Related Work

WSNs consist of small nodes with sensing, computation, and wireless communications capabilities. Many routing, power management, and data dissemination protocols have been specifically designed for WSNs where energy awareness is an essential design issue. Routing protocols in WSNs might differ depending on the application and network architecture [1]. Overall, the routing techniques are classified into three categories based on the underlying network structure: flat, hierarchical, and location-based routing. Furthermore, these protocols can be classified into multipath-based, query-based, negotiation-based, QoS-based and coherent based depending on the protocol operation.

Routing in WSNs is very challenging due to the inherent characteristics that distinguish these networks from other wireless ones like mobile ad-hoc or cellular networks. First, due to the relatively large number of sensor nodes, it is not possible to build a global addressing scheme for the deployment of a large number of sensor nodes as the overhead of ID maintenance is high. Thus, traditional IP-based protocols may not be applied to WSNs. Furthermore, sensor nodes that are deployed in an Ad-hoc manner need to be self-organizing as the Ad-hoc deployment of these nodes requires the system to form connections and cope with the resultant nodal distribution, especially as the operation of sensor networks is unattended. In WSNs, sometimes getting the data is more important than knowing the IDs of which nodes send it. Second, in contrast to typical communication networks, almost all applications of sensor networks require the fbw of sensed data from multiple sources to a particular BaseStation (BS). This, however, does not prevent the flow of data to be in other forms (e.g., multicast or peer to peer). Third, sensor nodes are tightly constrained in terms of energy, processing, and storage capacities and thus, they require careful resource management. Fourth, in most application scenarios, nodes in WSNs are generally stationary after deployment except for maybe a few mobile nodes. In other traditional wireless networks they are free to move, which results in unpredictable and frequent topological changes. However, in some applications, some sensor nodes

may be allowed to move and change their location (although with very low mobility). Fifth, sensor networks are application-specific (i.e., design requirements of a sensor network change with application). Sixth, position awareness of sensor nodes is important since data collection is normally based on the location. Currently, it is not feasible to use Global Positioning System (GPS) hardware for this purpose. Methods based on triangulation ([2]), for example, allow sensor nodes to approximate their position using radio strength from a few known points. It is found in [2] that algorithms based on triangulation or multilateration can work quite well under conditions where only very few nodes know their positions a priori (e.g., using GPS hardware). Still, it is favourable to have GPS-free solutions ([3]) for the location problem in WSNs. Finally, data collected by many sensors in WSNs is typically based on common phenomena, so there is a high probability that this data has some redundancy. Such redundancy needs to be exploited by the routing protocols to improve energy and bandwidth utilization. Usually, WSNs are data-centric networks in the sense that data is requested based on certain attributes (i.e., attribute-based addressing). An attribute-based address is composed of a set of attribute-value pair query.

Due to such differences, many new algorithms have been proposed for the routing problem in WSNs. These routing mechanisms have taken into consideration the inherent features of WSNs along with their application and architecture requirements. The task of finding and maintaining routes in WSNs is non-trivial since energy restrictions and sudden changes in node status (e.g., failure) cause frequent and unpredictable topological changes. To minimize energy consumption, routing techniques proposed in the literature, employ some well-known routing tactics as well as tactics special to WSNs, such as data aggregation and in-network processing, clustering, different node role assignment and data-centric methods.

Almost all of the routing protocols can be classified according to the network structure as flat, hierarchical, or location-based. Furthermore, these protocols can be classified into multipath-based, query-based, negotiation-based, quality of service (QoS)- based, and coherent-based depending on the protocol operation. In flat networks all nodes play the same role, while hierarchical protocols aim to cluster the nodes so that cluster heads can do some aggregation and reduction of data in order to save energy. Location-based protocols utilize position information to relay the data to the desired regions rather than the whole network ([4], [5]).

2.1 Routing Challenges and Design Issues in WSNs

Despite the innumerable applications of WSNs, these networks have several restrictions, such as limited energy supply, limited computing power, and limited bandwidth of the wireless links connecting sensor nodes. One of the main design goals of WSNs is to carry out data communication while trying to prolong the lifetime of the network and prevent connectivity degradation by employing aggressive energy management techniques. The design of routing protocols in WSNs is influenced by many challenging factors. These factors must be overcome before efficient communication can be achieved in WSNs. In the following, we summarize some of the routing challenges and design issues that affect the routing process in WSNs.

Node deployment: Node deployment in WSNs is application-dependent and can be either manual (deterministic) or randomized.

Energy consumption without losing accuracy: Sensor nodes can use up their limited supply of energy performing computations and transmitting information in a wireless environment. As such, energy-conserving forms of communication and computation are essential. Sensor node lifetime shows a strong dependence on battery lifetime [6]. In a multi-hop WSN, each node plays a dual role as data sender and data router. The malfunctioning of some sensor nodes due to power failure can cause significant topological changes, and might require re-routing of packets and reorganization of the network.

Data reporting method: Data reporting in WSNs is application-dependent and also depends on the time criticality of the data.

Node/link heterogeneity: In many studies, all sensor nodes were assumed to be homogeneous (i.e., have equal capacity in terms of computation, communication, and power). However, depending on the application a sensor node can have a different role or capability. The existence of a heterogeneous set of sensors raises many technical issues related to data routing. For example, some applications might require a diverse mixture of sensors for monitoring temperature, pressure, and humidity of the surrounding environment, detecting motion via acoustic signatures, and capturing images or video tracking of moving objects

Fault tolerance: Some sensor nodes may fail or be blocked due to lack of power, physical damage, or environmental interference. The failure of sensor nodes should not affect the overall task of the sensor network. If many nodes fail, medium access control (MAC) and routing protocols must accommodate formation of new links and routes to the data collection BSs.

Scalability: The number of sensor nodes deployed in the sensing area may be on the order of hundreds or thousands, or more. Any routing scheme must be able to work with this huge number of sensor nodes. In addition, sensor network routing protocols should be scalable enough to respond to events in the environment. Until an event occurs, most sensors can remain in the sleep state, with data from the few remaining sensors providing coarse quality.

Network dynamics: In many studies, sensor nodes are assumed fixed. However, in many applications both the BS or sensor nodes can be mobile [7]. As such, routing messages from or to moving nodes is more challenging since route and topology stability become important issues, in addition to energy, bandwidth, and so forth.

Connectivity: High node density in sensor networks precludes them from being completely isolated from each other. Therefore, sensor nodes are expected to be highly connected.

Coverage: In WSNs, each sensor node obtains a certain view of the environment. A given sensor's view of the environment is limited in both range and accuracy; it can only cover a limited physical area of the environment. Hence, area coverage is also an important design parameter in WSNs.

Data aggregation: Since sensor nodes may generate significant redundant data, similar packets from multiple nodes can be aggregated to reduce the number of transmissions. Data aggregation is the combination of data from different sources according to a certain aggregation function (e.g., duplicate suppression, minima, maxima, and average). This technique has been used to achieve energy efficiency and data transfer optimization in a number of routing protocols.

Quality of service: In some applications, data should be delivered within a certain

period of time from the moment it is sensed, or it will be useless. Therefore, bounded latency for data delivery is another condition for time-constrained applications.

2.2 Routing Protocols in WSNs

In general, routing in WSNs can be divided into flat-based routing, hierarchical-based routing, and location-based routing depending on the network structure. In flat-based routing, all nodes are typically assigned equal roles or functionality. In hierarchical-based routing, nodes will play different roles in the network. In location-based routing, sensor nodes positions are exploited to route data in the network. A routing protocol is considered adaptive if certain system parameters can be controlled in order to adopt to current network conditions and available energy levels. Furthermore, these protocols can be classified into multipath-based, query-based, and negotiation-based, QoS-based, or coherent-based routing techniques depending on the protocol operation. In addition to the above, routing protocols can be classified into three categories, proactive, reactive, and hybrid, depending on how the source finds a route to the destination. In proactive protocols, all routes are computed before they are really needed, while in reactive protocols, routes are computed on demand. Hybrid protocols use a combination of these two ideas. When sensor nodes are static, it is preferable to have table-driven routing protocols rather than reactive protocols. A significant amount of energy is used in route discovery and set-up of reactive protocols. Another class of routing protocols is called cooperative. In cooperative routing, nodes send data to a central node where data can be aggregated and may be subject to further processing, hence reducing route cost in terms of energy use. Many other protocols rely on timing and position information. In the following subsections we present a classification according to the network structure and protocol operation (routing criteria).

2.2.1 Network-Structure-Based Protocols

The underlying network structure can play a significant role in the operation of the routing protocol in WSNs.

2.2.1.1 Flat-Routing

The first category of routing protocols are the multi-hop flat routing protocols. In flat networks, each node typically plays the same role and sensor nodes collaborate to perform the sensing task. Due to the large number of such nodes, it is not feasible to assign a global identifier to each node. This consideration has led to data-centric routing, where the BS sends queries to certain regions and waits for data from the sensors located in the selected regions. Since data is being requested through queries, attribute-based naming is necessary to specify the properties of data. Early work on data centric routing (e.g., SPIN and directed diffusion [8]) were shown to save energy through data negotiation and elimination of redundant data. These two protocols motivated the design of many other protocols that follow a similar concept. Some of these protocols are the following:

- Sensor Protocols for Information via Negotiation (SPIN) ([9], [10], [11])
- Directed diffusion ([12])
- Rumor Routing ([13])
- Minimum Cost Forwarding ([8])
- Gradient-based routing ([14])
- Information-driven sensor querying and constrained anisotropic diffusion routing ([15])
- GOUGAR ([16])
- ACQUIRE ([17])
- Energy-Aware Routing ([18])
- Routing Protocols with random walks ([19])

2.2.1.2 Hierarchical Routing

Hierarchical or cluster based routing methods, originally proposed in wire line networks, are well-known techniques with special advantages related to scalability and efficient communication. As such, the concept of hierarchical routing is

also utilized to perform energy-efficient routing in WSNs. In a hierarchical architecture, higher-energy nodes can be used to process and send the information, while low-energy nodes can be used to perform the sensing in the proximity of the target. The creation of clusters and assigning special tasks to cluster heads can greatly contribute to overall system scalability, lifetime, and energy efficiency. Hierarchical routing is an efficient way to lower energy consumption within a cluster, performing data aggregation and fusion in order to decrease the number of transmitted messages to the BS. Hierarchical routing is mainly two-layer routing where one layer is used to select cluster heads and the other for routing. However, most techniques in this category are not about routing, but rather who and when to send or process/ aggregate the information, channel allocation, and so on, which can be orthogonal to the multihop routing function. Some of these protocols are enumerated below:

- LEACH protocol ([6])
- Power-Efficient Gathering in Sensor Information Systems ([20])
- Threshold-Sensitive Energy Efficient Protocols ([21], [22])
- Small minimum energy communication network (MECN) ([23])
- Sensor aggregates routing ([24])
- Hierarchical power-aware routing ([25])
- Two-Tier Data Dissemination ([7])

2.2.1.3 Location-Based Routing Protocols

In this kind of routing, sensor nodes are addressed by means of their locations. The distance between neighboring nodes can be estimated on the basis of incoming signal strengths. Relative coordinates of neighboring nodes can be obtained by exchanging such information between neighbors ([2], [3], [25]). Alternatively, the location of nodes may be available directly by communicating with a satellite using GPS if nodes are equipped with a small low-power GPS receiver. To save energy, some location-based schemes demand that nodes should go to sleep if there is no activity. More energy savings can be obtained by having as many sleeping nodes in the network as possible. The problem of designing sleep period schedules for each node in a localized manner was addressed in [26].

2.2.2 Routing Protocols based on Protocol Operation

2.2.2.1 Multipath Routing Protocols

In this subsection we study routing protocols that use multiple paths rather than a single path in order to enhance network performance. The fault tolerance (resilience) of a protocol is measured by the likelihood that an alternate path exists between a source and a destination when the primary path fails. This can be increased by maintaining multiple paths between the source and destination at the expense of increased energy consumption and traffic generation. These alternate paths are kept alive by sending periodic messages. Hence, network reliability can be increased at the expense of increased overhead in maintaining the alternate paths. Such protocols are presented in [27], [18], [28] and [29].

2.2.2.2 Query-Based Routing

In this kind of routing, the destination nodes propagate a query for data (sensing task) from a node through the network, and a node with this data sends the data that matches the query back to the node that initiated the query. Usually these queries are described in natural language or high-level query languages. An example of a Query-Based routing protocol is rumor routing protocol ([13]), which uses a set of long-lived agents to create paths that are directed towards the events they encounter.

2.2.2.3 Negotiation-Based Routing Protocols

These protocols use high-level data descriptors in order to eliminate redundant data transmissions through negotiation. Communication decisions are also made based on the resources available to them. The SPIN family protocols [9] discussed earlier and the protocols in [10] are examples of negotiation-based routing protocols.

2.2.2.4 QoS-based Routing

In QoS-based routing protocols, the network has to balance between energy consumption and data quality. In particular, the network has to satisfy certain QoS metrics (delay, energy, bandwidth, etc.) when delivering data to the BS. [30]

2.2.2.5 Coherent and Non-coherent Processing

Data processing is a major component in the operation of wireless sensor networks. Hence, routing techniques employ different data processing techniques. In general, sensor nodes will cooperate with each other in processing different data flooded in the network area.

Table 2.1: Hierarchical vs. Flat topologies Routing

Hierarchical Routing	**Flat Routing**
Reservation-based scheduling	Contention-based scheduling
Collisions avoided	Collision Overhead present
Reduced duty cycle due to periodic sleeping	Variable duty cycle by controlling sleep time of nodes
Data aggregation cluster-head	Node on multi-hop path aggregates incoming data from neighbours
Simple but non-optimal routing	Routing optimal with added complexity
Global and local synchronization	Links formed on the fly without sync
Energy dissipation is uniform	Energy dissipation depends on traffic patterns

2.3 Trust Information Communication Models

A Wireless Sensor Network should satisfy a number of security requirements. An important subset of the security threats can be addressed implementing a trust management system, which establishes trust relationships among the network nodes. The methods for calculating trust via concatenation and multipath propagation are referred to as trust models. These relationships are then taken into account when a node needs the collaboration of its neighbours to accomplish a task, such as routing and data aggregation. While key-based techniques can be used to provide data integrity and confidentiality, a trust model is required for higher layer decisions, which are critical in this collaborative networking environment formed by the sensor networks. The output of this scheme will be a single trust value that will be taken into account when deciding the routing path. Trust schemes work towards the effective increase of availability and reliability of a sensor network which is of primary concern taken into account the unattended and infra-structureless nature of such networks. For a complete estimation of such a scheme not only computational

complexity but also the level of increase of the total network throughput and the impact on energy consumption must be thoroughly examined in order to evaluate its effective applicability in WSN networks.
Unlike infrastructure-based networks, nodes need to cooperate in order to route packets in ad-hoc and sensor networks. However, in this unmanaged environment, trusting the neighbor for forwarding the traffic is not a wise option. To establish trust relationships, the nodes are required to evaluate the trust of their neighbors and communicate in order to exchange trust information, exactly as happens in every society. To evaluate the trust, each node monitors the behaviour of its neighbors and/or challenges it. The outcome may be used by itself or can be announced to other nodes as well.
In this chapter are investigated various trust models focusing on the way trust information is communicated among nodes, i.e. we will study how the nodes perceive their neighbors trust.

2.3.1 Basic Notions of Trust Definition

Before the analysis of the different aspects and characteristics of WSN trust models a short description for the meaning of trust is provided in this section. Trust concept has been used in the literature for many purposes. For example, authors in [31] present a new approach of dynamic symmetric key distribution algorithm for encrypting the communication between two nodes in a wireless sensor network, by using a localized node computation of a trust value granted by the requesting nodes, without the need for a trusted third party node. Below we provide some necessary definitions:

- **Trust** is defined as the confidence of an entity on another entity based on the expectation that the other entity will perform a particular action important to the truster, irrespective of the ability to monitor or control that other entity.
- **Reputation** is the opinion of one entity about another and in an absolute context it is the trustworthiness of an entity.

Trust establishment can be considered as the application of an evaluation metric to a body of trust evidence that is constructed from the set of attributes that define trust. The outcome of the trust establishment process is a trust relation.

Trust relations are based on evidence created by the previous interactions of entities within a network. The evidence may be obtained on- or off-line. An established trust relation can be used in other trust establishment processes and can be combined with other relations to form more general trust relations. In [32] Trust Establishment is defined as the specification of admissible types of trust evidence, the generation, distribution, discovery and evaluation of trust evidence. Trust establishment can be considered as a process which involves the following steps:

1. Collection of evidence from network entities
2. Evaluation of trust based on a defined set of metrics
3. Propagation of trust evaluation results in other network entities

2.3.2 Trust Models and Classification

To thoroughly study trust models, we need to identify the building blocks they consist of, or the invoked processes. There are several options to be decided when designing a trust model. To provide a preview:

- The participating entities monitor the behavior of neighboring nodes, in order to obtain trust-related data. The aspects of behavior under measurement may vary, from model to model as well as the way this is quantified. The measurement may be performed by one, more or all the nodes of the network. Each monitoring (source) node may monitor/observe the behavior of one, a set (possibly the nodes of its cluster) or all the neighboring nodes of the network.
- Once the trust data are formed, they are communicated, either to the upper layers of the node (in which case the trust values are consumed by the node itself and the model is based on direct observations only) or to other nodes as well, building a reputation scheme. In the later case, the trust information may travel towards any node requesting this information, all the neighboring nodes (flooding of trust information), nodes fixed number of hops away (limited flooding), a set of nodes (aggregator nodes, or nodes that have used this path recently) or a single trusted node (sink) or cluster head ([33]).
- The trust information is used to decide the eligibility of a node for a specific task, or the exclusion of the node from candidates. The criteria for this choice

is usually the overall node trust value (which may be a function of different trust metrics) while the selected threshold values depend on the desired sensitivity. Other response options are defensive response, offensive response, and dismissal response ([34]). A defensive response avoids using bad reputed nodes but accepts their packets. In an offensive response, a node refuses to forward packets received from an observed node that violates the trust limits, punishing it. With an extreme violation of trust limits, a bad node is totally dismissed with zero chance to be accepted.

- The eligibility decision may be taken in one trusted node only (sink or cluster head) or can be taken in a distributed way either from all nodes in the network during their routing decisions or by part of them. When the trust information is used to decide the non-trusted nodes,these decisions may travel to other nodes based again on some announcement protocol which defines whether this information will be communicated to all nodes, or to a list of nodes or to single nodes upon request.

A variety of trust models will be evaluated along the above four directions ranging from proactive and reactive ad-hoc routing protocols [35] (e.g. DSR, AODV, TORA) to trust-modified geographic routing protocols (GPSR, GEAR). We will try to organize them in categories but mainly evaluate them. Different categories of trust models can be formulated based on each arising option described above, as also stated in [32]. For example, we can distinguish trust models which are based on direct trust measurements from others which are based on both direct and indirect trust measurement. We can also classify the trust models based on the network element that measure the trust which can be one, a set of special nodes or all of them. Another option is to classify them according to the node that decides the trusted/non trusted nodes. In the sequence, we classify the trust models in three categories: centralized, hierarchical and distributed based on the split of functionality among the participating nodes.

2.3.3 Centralized Approach

In a centralized trust model, a (head) node undertakes the responsibility to decide the node's trustworthiness, based either on trust data it has collected on its own, or on trust data received by other nodes. The head node issues a broadcast

message requesting trust information and all the receiving nodes respond with the trust data. Then these data are used to select the most trustful node to undertake a certain task, which is usually the forwarding task. The decision is then announced back to the participating nodes, to appropriately update the relevant information. In [35], a similar scheme is adopted to elect the next cluster head in the cluster. The current cluster head undertakes the responsibility to gather trust information and decide the next cluster head. This protocol is suitable for sensor network which follow clustering architectures or assign a node with a specific task. This task can be data forwarding (e.g. a node with higher capabilities may be assigned the task of forwarding the packets of the cluster) or data aggregation or even trust evaluation, as for example in [36] the nodes calculate their neighbors trust based on the information received from an "anchor" apart from their own observations. These nodes become then more important than others and represent potential single points of failure, thus necessitating the implementation of a trust model [37].

2.3.4 Hierarchical Approach

To economize resources such as transmission power and bandwidth, dense sensor networks can be are divided in groups/clusters and one (or more) node in each cluster undertakes special responsibility. The division can be based on location or application criteria. For example, nodes sensing the temperature can form a group while nodes for other physical parameters may belong to a different group. Special responsibilities include (but are not limited to) forwarding towards the sink or data aggregation or trust measurement. Different solutions for choosing the "special" nodes exist (such as the LEACH scheme) [6]. Here, we focus on trust models designed for sensor networks of hierarchical structure.

2.3.5 Distributed Approach

Trust models assume that no sensor has been assigned a specific role in the trust management system. Direct trust measurements are used as a criterion for routing purposes while a combination of direct and indirect measurements can also be taken into account. Although the indirect measurements could be considered only as a component of the trust value, actually they play a more significant role than that. They are part of the trust system management design since the communication of this information introduces the need for a specific protocol (type of messages

exchanged,frequency, interactions, etc.). Moreover, they affect the performance of the sensor network since they consume resources for transmission and processing.

2.3.6 Hybrid Schemes

Beyond completely centralized or distributed trust management schemes, hybrid schemes compromising resource consumption for achieved security have also been reported. An example is the "Group - based trust management" scheme proposed in [38]. This is a novel lightweight group based trust management scheme for distributed wireless sensor networks in which the whole group gets a single trust value. Within the group the distributed trust management approach is used in which all sensor nodes need to calculate individual trust values for all group members. Cluster head is responsible for trust values aggregation and forwarding to the base station. Depending upon the trust values concentrated in base station each whole group is assigned one out of three possible values: trusted, un-trusted and uncertain.

2.3.7 Trust Model Attacks and Countermeasures

Trust management can effectively improve network performance. Therefore, trust management itself is an attractive target for attackers. For example, trust evaluation systems also suffer from sybil attacks when a malicious node can create faked IDs that share or even take the blame which should be given to the malicious node. Another possible attack is the newcomer's attack: a malicious node removes its bad history by registering as a new user. The defence against the Sybil attack and newcomer attack does not rely on the design of trust evaluation, but the authentication schemes. Other attacks that threaten trust evaluation systems include:

- Bad Mouthing Attack
- On-Off Attack
- Conflicting Behavior Attack

2.4 Trust Metrics Identification and Trust Value Derivation

The actions expected to be taken by every node in a sensor network include forwarding of packets, appropriate handling of routing messages, sincere participation in the network operation, performing forwarding, aggregation and specific protocol-related information communication etc. Each action can be described using a set of different attributes and thus each node can be described by its ID, the actions it contributes in and the relevant attributes values. When the cooperation process among the nodes finishes, the trust relationship is set up among the nodes, and the historical cooperation records can be built gradually. Trust can be build monitoring different aspects of the behavior of the neighboring node.

2.4.1 Metrics for Trust Establishment in Ad-Hoc Sensor Networks

There is no consensus among researchers on the behavior aspects that should be monitored in order to calculate a single total trust value for each node. This is is not surprising since the adopted metrics depend on the type of attack the network should be shielded from. The type of monitored aspect affects the types of attack the trust system can face. For example, if the message integrity is not monitored, any attack based on message modification will not be combated.
For this reason, in this section, we investigate the node behaviors that have to be monitored in order to define its trustworthiness and then we study how this information can be used to calculate a trust value. Since monitoring multiple behavior aspects and the maintenance of the relevant information consumes valuable resources (in terms of power, processing capability and memory), the target is to identify the metrics that bring the best result with the available resources.

- **Data Packets Forwarded**: Each node in the network buffers every transmitted packet for a limited period. During this time, each node places its wireless interface into promiscuous mode in order to overhear whether the next node has forwarded the packet or not. The information collected through this scheme is then used by the Pathrater entity in order to define different (dynamically varying) ratings for the neighboring nodes. Watchdog and Pathrater have been designed to enrich with trust the Dynamic Source Routing protocol (which is a re-active routing protocol), although they are also applicable to a wide range of routing protocols.

- **Control Packets Forwarded**: Similarly to the observations regarding the data packet forwarding, the forwarding of control (routing) messages can be monitored in order to evaluate the trust of a sensor node. The method of passive acknowledgement can be further classified into acknowledgements for data packets and acknowledgements for control packets, to produce two different trust metrics. This is meaningful only if we intend to assign different weight values to each metric. The approach is similar to the one described for data packets: for each transmitted control packet a counter is maintained and the source node listens the transmissions of the destination node in order to check whether the control message has been forwarded. In case, data forwarding is distinguished from control messages forwarding, a different set of counter is needed. The trust related to control frames is expressed similarly to the data forwarding metric [39].

- **Data Packet Precision**: The forwarding of altered packets should raise an alarm and the node should be marked from suspicious to malicious. This way all attacks related to message modifications can be combated. This metric is applicable to all routing schemes. The category Packet Precision [40] ensures the integrity of the data and control packets that are either received or forwarded by other nodes in the network.

- **Other metrics**: "Hello" replies measurements [37], Packet Address Modified [40], Cryptography, Correct routing protocol execution [39].

2.4.2 Trust Quantification and Trust Computation

The trust value represents a total trustworthiness of a node, which is evaluated based on a set of trust factors expressing the metrics described in the previous section. The trust value of a node changes dynamically because the values of each trust evaluation factor change with the lapse of time.

2.4.2.1 Trust Value Calculation

Different options regarding the calculation of the overall trust value exist: it can be calculated as a product of trust-related parameters value or as a weighted sum where each trust parameter is assigned a different weight, i.e. level of importance.

The case of product has been initially proposed in [37] where three different trust parameters were used: the Cryptography capability of the Node (Ci), the Availability (based on beacon messages) and the Packet forwarding. The equation expressing the trust value is:

$$T_i = C_i * A_i * b * P_i \tag{2.1}$$

It is worth stressing that although A_i and P_i range from 0 to 1, two values are only possible for C_i 1 and 0, which emphasizes on the value of cryptography in securing the network communication. An encouragement factor b was also introduced in calculating the forwarding trust value which is initially set high and decreases with increase in number of samples to encourage newly activated nodes to forward packets.

The initial trust values have attracted a lot of attention, with the most popular proposals assigning an average trust value, when needed. No such choice is required in the case that packet forwarding is expressed as the ratio of successfully forwarded packets over the received packets, assuming that the trusted routing protocol does not use the maximum trust value criteria for routing. If this assumption does not hold, new nodes will never get the opportunity to forward a packet unless every other node is exhausted.

An approach widely adopted defines the overall trust as a sum of weighted factors. This means that the trust parameters are first the significance of the measurement, i.e.

$$T_{xy} = \sum_{i=1}^{n} [W_{xy}(i) * T_{xy}(i)] \tag{2.2}$$

Where T_{xy} is the situational trust $T_{xy}(i)$ in node y calculated by node x for trust category i, n the number of evaluated metrics and $W_{xy}(i)$ the weight assigned to the trust metric i.

Slightly different proposals for the calculation of the overall trust value have been defined in [40].

2.4.2.2 Statistical Trust Calculation

Other approaches in trust calculation have concentrated in statistical methods ([41]). In [41] the output of the trust mechanism is a trust value and a confidence interval around this value based on direct and indirect experiences of sensor node behavior. Statistical values are used both in initial evaluation of experience records

as well as the collected experiences by third parties. An assumption taken in this work is the existence of sufficient redundancy in the amount of sensor nodes to meet the application needs. The correctness of sensed data values can be verified by close neighbors. In the initial evaluation the value of confidence interval is used to determine the existence of sufficient evidence in order to take the decision if the node can be considered trusted or not.

2.5 WSN Reconfigurable Platforms

One of the most important features of a successful WSN platform is its low energy consumption. That is the main reason that most available WSN nodes include a bare minimum of circuitry typically consisting of some sensoring devices, a simple micro controller and a simple wireless communication module.
Recent advances in micro-electronics, and especially in the field of reconfigurable devices, have enabled the development of reconfigurable, low-cost, low-power and high performance devices that can be deployed in WSN nodes. In this section, some of these platforms are presented, which are, according to our knowledge, the first attempts to introduce reconfigurable devices in the field of WSNs.

2.5.1 mPlatform

mPlatform [42] is a new reconfigurable modular sensornet platform that enables real-time processing on multiple heterogeneous processors. The heart of the mPlatform is a scalable high-performance communication bus connecting the different modules of a node, allowing time-critical data to be shared without delay and supporting reconfigurability at the hardware level. Furthermore, the bus allows components of an application to span across different processors/modules without incurring much overhead, thus easing the program development and supporting software reconfigurability. In [42], the authors describe their communication architecture and protocol as well as their implementation in a low power, high speed complex programmable logic device (CPLD). An asynchronous interface decouples the local processor of each module from the bus, allowing the bus to operate at the maximum desired speed while letting the processors focus on their real-time tasks such as data collection and processing. Extensive experiments on the mPlatform prototype have validated the scalability of the communication architecture,

and the high bandwidth provided at the expense of a small increase in the power consumption. Finally, they demonstrate a realtime sound source localization application on the mPlatform, with four channels of acoustic data acquisition, FFT, and sound classification, that otherwise would be infeasible using traditional buses such as I2C.

mplatform addresses the inter-module communication problem by introducing a

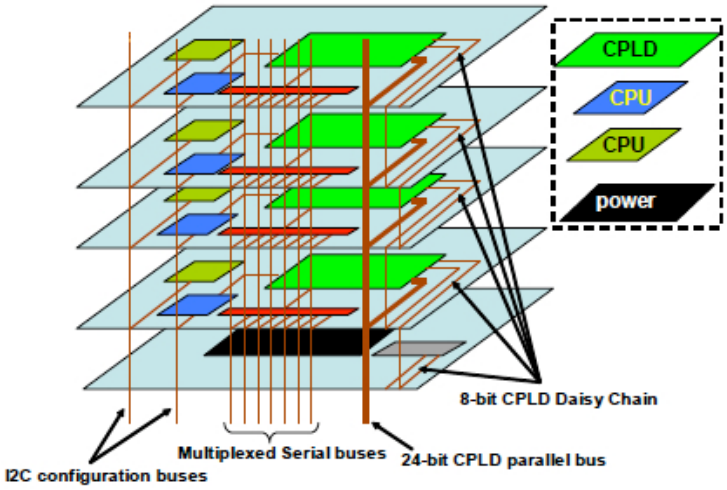

Figure 2.1: Overview of the mPlatform architecture.

new flexible, efficient and reconfigurable communication channel architecture that better fits the needs of modular sensor network platforms. The architecture is based on the following key design requirements: *Resource efficiency, Processor Independence, Scalability,Fairness, Reconfigurability.*

mPlatform meets these design requirements by abstracting the communication channel from the communicating processors. The local processor on each module interacts with a parallel bus through a bus controller, implemented in a low-power CPLD. This approach decouples the communication channel from the local processor, allowing different processors running at different speeds to share the bus without impacting its throughput. To guarantee fairness a TDMA-based protocol implemented in the bus controller allows multiple processors to exchange data al-

most simultaneously. The high data rate enabled by the parallel bus combined with the TDMA protocol create a near real time inter-module communication channel that scales well with the number of modules in the stack. Processor independence and resource efficiency are achieved by enforcing an asynchronous interface over a separate parallel bus between the CPU and the bus controller. This enables the bus controller to be transparently interfaced to a processor running at any speed. The asynchronous nature of the interface enables the processor to transfer data at a speed usually limited by the processor clock speed because of the relatively high clock speed of the bus controller.

The advanced functionality of the new communication architecture comes at the expense of a small increase in the power consumption of the platform mainly due to the use of CPLD. However, the flexibility afforded by the CPLD outweighs the small power overhead. The mPlatform architecture is a research platform designed to facilitate rapid prototyping and experimentation. Complex programmable logic devices provide an abstraction of the hardware layer that drastically simplifies the tinkering at the protocol level requiring little change to hardware.

mPlatform is a modular architecture that focuses on providing a Lego-like, plug-and-play capability to put together a sensor network platform tailored to specific application/research requirements. The hardware architecture design was driven by the following basic guidelines:

Reconfigurability: The architecture needs to be easily reconfigurable to meet specific needs of a particular research project. For example, a data collection task with a low sampling rate may just require an 8-bit processor and a slow radio connection to a gateway to conserve power, while a physiological monitoring application on a body sensor network that alerts a remote physician upon detecting an abnormal condition will need more processing power to analyze the signals, enough storage for disconnected operation, and the ability to connect to multiple wireless networks. To enable reconfigurability, mPlatform was designed so that a wide range of processors, from MSP430 class processors up to PXA270 processors, can co-exist on the same platform and efficiently communicate in any possible configuration.

Real-time event handling: Since a sensor platform is typically used in applications where it constantly interacts with the environment, the ability to handle real-time events is crucial. Examples include detection of an abnormally high temperature indicating a fire, detection of an abnormal physiological signal, and arrival of a radio packet.

Fine-grained power management: In many sensing and mobility applications nodes are powered by battery or salvaged energy sources. It is desirable to be able to shut down components when not in use and scale up or down operation voltage and/or frequency of the components in order to accommodate task needs while conserving energy and other resources.

2.5.2 Atific Helicopter

The design and implementation of an innovative high performance multi-radio WSN platform is presented in [43], [44]. This platform includes both multi-processor devices and extremely flexible and fully re-configurable FPGAs. The use of four parallel radio transceivers with 83 selectable frequency channels allows the development of communication protocols with high interference tolerance, low latency and high mesh-networking performance. Purely simultaneous data exchanges with several neighbors are possible. Case studies present the performance of a number of multi-processor implementations including from 1 to 4 parallel Altera [45] Nios II [46] soft core processors. In addition, two design cases achieving ultra low latency and high network throughput employing up to four radio transceivers are also presented.
The platform is targeted for demanding WSN applications for surveillance and control in in-door environment. The platform operates in large mesh networks, where high network throughput and security, very low routing delay, and very high interference tolerance are required.
The use of FPGA enables timely accurate parallel processing, reconfigurability and high processing performance. The processing architecture can be changed in field after deployment.

The hardware architecture of the platform is presented in Figure 2.2. All the

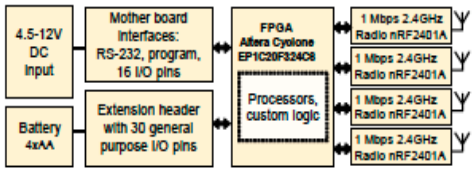

Figure 2.2: Atific Platform Hardware Architecture

processing takes place on Altera Cyclone EP1C20 [47] FPGA. The FPGA is large enough for implementing several embedded processor cores, system memory, and custom hardware accelerators. The Cyclone FPGA contains 20,060 Logic Elements (LEs) and 294,912 bits of embedded dual-port RAM. These resources are suited for implementing versatile multi-processor architectures. Further, custom hardware accelerators can be synthesized into the FPGA for accelerating functions that are performed faster and with lower energy on dedicated logic than on a processor.

Normally the FPGA configuration is loaded from an on-board flash memory on

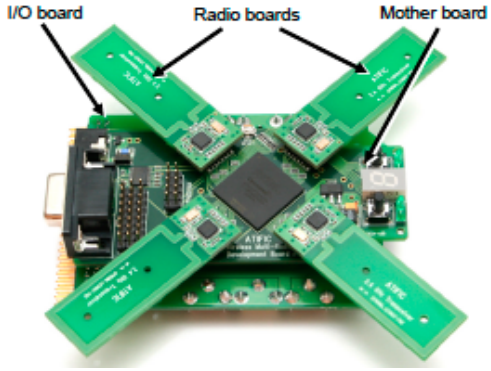

Figure 2.3: Atific Platform Top View

power up. Alternatively, Altera download cable can be used to configure both the flash memory and FPGA.

Wireless communication use four Nordic Semiconductor nRF2401A [48] radio transceivers. The radios have 1 Mbps data rate and 83 selectable frequency channels in the 2.4000 2.4835 GHz license-free frequency band (Europe region). Thus, a locally unique frequency channel may be assigned for each radio link. This may be utilized for developing very high performance link layer protocols.

The platform can operate as stand-alone node in the WSN, but it can also be connected to a PC computer for debugging and monitoring of internal operation. This is enabled by RS-232 serial port [49]. PC can also accommodate some of the WSN application or protocol functionality especially during the development phase.

The complete WSN implemented platform is presented in Figure 2.3. This platform, according to the authors, gives excellent opportunities to design and test new kind of WSN protocols, algorithms and applications.

Chapter 3

GPSR Routing Protocol and XMesh Protocol Stack

In this chapter we present the Greedy Perimeter Stateless Routing (GPSR) [50], [51], [52], [53] a novel routing protocol for wireless datagram networks that uses the positions of routers and a packet's destination to make packet forwarding decisions. GPSR makes greedy forwarding decisions using only information about a router's immediate neighbours in the network topology. When a packet reaches a region where greedy forwarding is impossible, the algorithm recovers by routing around the perimeter of the region. By keeping state only about the local topology, GPSR scales better in per-router state than shortest-path and ad-hoc routing protocols as the number of network destinations increases. Under mobility's frequent topology changes, GPSR can use local topology information to find correct new routes quickly.
Later in this chapter we present also a Trust model for the GPSR protocol, that makes it secure and trustworthy. The outcome of this model will be the forwarding trustworthiness of each node and will be exploited to define secure routes to the destination.
Finally, we present another WSN routing protocol, the XMesh protocol stack, which is a widely used multihop routing protocol, famous for its performance when applied in real world WSNs.

3.1 Greedy Perimeter Stateless Routing

In networks comprised entirely of wireless stations, communication between source and destination nodes may require traversal of multiple hops, as radio ranges are finite. A community of ad-hoc network researchers has proposed, implemented, and measured a variety of routing algorithms for such networks and they concluded to the observation that topology changes more rapidly on a mobile, wireless network than on wired networks, where the use of Distance Vector (DV), Link State

(LS), and Path Vector routing algorithms is well established. DV and LS algorithms require continual distribution of a current map of the entire network's topology to all routers. DV's Bellman-Ford approach constructs this global picture transitively; each router includes its distance from all network destinations in each of its periodic beacons. LS's Dijkstra approach directly floods announcements of the change in any link's status to every router in the network. Small inaccuracies in the state at a router under both DV and LS can cause routing loops or disconnection [54]. When the topology is in constant flux, as under mobility, LS generates torrents of link status change messages, and DV either suffers from out-of-date state [55], or generates torrents of triggered updates.

The two dominant factors in the scaling of a routing algorithm are: the rate of change of the topology and the number of routers in the routing domain.

Both factors affect the message complexity of DV and LS routing algorithms: intuitively, pushing current state globally costs packets proportional to the product of the rate of state change and number of destinations for the updated state.

Hierarchy is the most widely deployed approach to scale routing as the number of network destinations increases. Without hierarchy, Internet routing could not scale to support today's number of Internet leaf networks. An Autonomous System runs an intra-domain routing protocol inside its borders, and appears as a single entity in the backbone inter-domain routing protocol, BGP. This hierarchy is based on well-defined and rarely changing administrative and topological boundaries. It is therefore not easily applicable to freely moving ad-hoc wireless networks, where topology has no well-defined AS boundaries, and routers may have no common administrative authority.

Caching has come to prominence as a strategy for scaling ad-hoc routing protocols. Dynamic Source Routing (DSR) [56], Ad-Hoc On-Demand Distance Vector Routing (AODV) [57], and the Zone Routing Protocol (ZRP) [58] all eschew constantly pushing current topology information network-wide. Instead, routers running these protocols request topological information in an on-demand fashion as required by their packet forwarding load, and cache it aggressively. When their cached topological information becomes out-of-date, these routers must obtain more current topological information to continue routing successfully. Caching reduces the routing protocols' message load in two ways: it avoids pushing topological information where the forwarding load does not require it (e.g., at idle routers), and it often reduces the number of hops between the router that has the needed topological in-

formation and the router that requires it (i.e., a node closer than a changed link may already have cached the new status of that link).
The aggressive use of geography allows GPSR to achieve scalability. Geographic routing allows routers to be nearly stateless, and requires propagation of topology information for only a single hop: each node need only to know its neighbors positions. The self-describing nature of position is the key to geography's usefulness in routing. The position of a packet's destination and positions of the candidate next hops are sufficient to make correct forwarding decisions, without any other topological information.

3.1.0.1 Greedy Forwarding

We now describe the first part of the Greedy Perimeter Stateless Routing algorithm: greedy forwarding. In this chapter, we define the greedy forwarding rule; define a simple beaconing protocol for nodes to learn their neighbors' positions; identify the desirable properties of greedy forwarding; define the topologies on which greedy forwarding fails and characterize the frequency of greedy forwarding failure by the density of nodes in a network.
Under GPSR, packets are marked by their originator with their destinations' locations. As a result, a forwarding node can make a locally optimal, greedy choice in selecting a packet's next hop. Specifically, if a node knows its radio neighbors' positions, the locally optimal choice of next hop is the neighbor geographically closest to the packet's destination. Forwarding in this regime follows successively closer geographic hops, until the destination is reached. An example of greedy next-hop choice appears in Figure 3.1. Here, x receives a packet destined for D. X's radio range is denoted by the dotted circle about x, and the arc with radius equal to the distance between y and D is shown as the dashed arc about D. x forwards the packet to y, as the distance between y and D is less than that between D and any of x's other neighbors. This greedy forwarding process repeats until the packet reaches D.
Suppose the header of a packet p contains fields p.a, the address of the packet's destination, and p.l, the location of the packet's destination. Moreover, assume for the moment that each node has a neighbor table N, each of whose entries is a pair of a neighbor node's address (a) with that neighbor's location (l). We denote a node's own address and location by self.a and self.l.

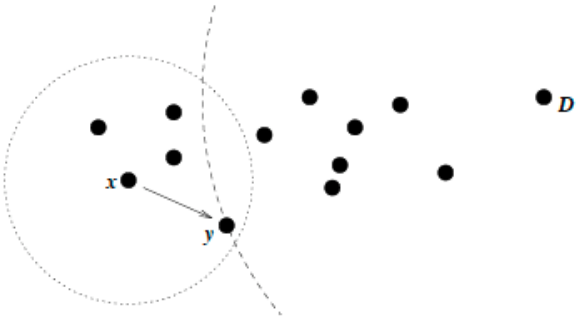

Figure 3.1: Greedy forwarding example. y is x's closest neighbor to D.

A simple beaconing protocol provides all nodes with their neighbors' positions: periodically, each node transmits a beacon to the broadcast MAC address, containing only its own identifier (e.g., IP address) and position. Position is encoded as two four-byte floating-point quantities, for x and y coordinate values. To avoid synchronization of neighbors' beacons, as observed by Floyd and Jacobson [59], each beacon's transmission is jittered by 50% of the interval B between beacons, such that the mean inter-beacon transmission interval is B, uniformly distributed in [0.5B, 1.5B].

Upon not receiving a beacon from a neighbor for longer than timeout interval T, a GPSR router assumes that the neighbor has failed or gone out-of-range, and deletes the neighbor from its table. The 802.11 MAC layer also gives direct indications of link-level retransmission failures to neighbors.

The position a node associates with a neighbor becomes less current between beacons as that neighbor moves. The accuracy of the set of neighbors also decreases; old neighbors may leave and new neighbors may enter radio range. For these reasons, the correct choice of beaconing interval to keep nodes' neighbor tables current depends on the rate of mobility in the network and range of nodes' radios.

This beaconing mechanism does represent pro-active routing protocol traffic, avoided by DSR and AODV. To minimize the cost of beaconing, GPSR piggybacks the local sending node's position on all data packets it forwards, and runs all nodes' network interfaces in promiscuous mode, so that each station receives a copy of all packets for all stations within radio range. At a small cost in bytes, this scheme allows all packets to serve as beacons. When any node sends a data packet, it can then reset

its inter-beacon timer. This optimization reduces beacon traffic in regions of the network actively forwarding data packets.

Greedy forwarding's great advantage is its reliance only on knowledge of the forwarding node's immediate neighbors. The state required is negligible, and dependent on the density of nodes in the wireless network, not the total number of destinations in the network [1]. On networks where multi-hop routing is useful, the number of neighbors within a node's radio range must be substantially less than the total number of nodes in the network.

The density in space of the nodes deployed on a wireless network increases, greedy forwarding approximates shortest paths progressively more closely; the shortest path between two nodes tends towards the Euclidean straight line between them, as the minimum possible number of hops is bounded below by the number of radio ranges between source and destination, laid end-to-end.

Traditional shortest-path routing algorithms cannot exploit structure in IP addresses to make forwarding decisions; they must treat IP addresses as flat identifiers, and resort to a table lookup among all destinations in the routing domain. It is the self-describing nature of geographic co-ordinates that allows forwarding routers to interpret the destination location in a packet to make a purely local forwarding decision.

Note that the only routing protocol traffic required for greedy forwarding is that of the beaconing protocol. Because the beaconing protocol pushes state only a single hop in the network, intuitively it should consume considerably less bandwidth than protocols which distribute state globally throughout the routing domain (e.g., DV and LS routing protocols), or accumulate state along an entire source route (e.g., DSR).

Because greedy forwarding makes purely local decisions, it should be robust under topological changes; a node can make correct forwarding decisions without requiring up-to-date state (or indeed, any state) concerning nodes beyond a single hop away.

The power of greedy forwarding to route using only neighbor nodes' positions comes with one attendant drawback: there are topologies in which the only route to a destination requires a packet move temporarily farther in geometric distance from the destination. A simple example of such a topology is shown in Figure 3.2. Here, x is

[1] The word "stateless" in GPSR's name is not meant literally, but refers to this small, purely local state.

closer to D than its neighbors w and y. Again, the dashed arc about D has a radius equal to the distance between x and D. Although two paths, $(x \rightarrow y \rightarrow z \rightarrow D)$ and $(x \rightarrow y \rightarrow w \rightarrow D)$ exist to D, x will not choose to forward to w or y using greedy forwarding. x is a local maximum in its proximity to D. Some other mechanism must be used to forward packets in these situations.

Motivated by Figure 3.2, we note that the intersection of x's circular radio range and the circle about D of radius $|\overline{xD}|$ (that is, of the length of line segment $\overline{xD}$) is empty of neighbors. This region is clearly presented in Figure 3.3. From node x's perspective, we term the shaded region without nodes a void. x seeks to forward a packet to destination D beyond the edge of this void. Intuitively, x seeks to route around the void; if a path to D exists from x, it doesn't include nodes located within the void (or x would have forwarded to them greedily).

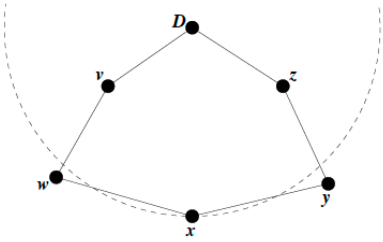

Figure 3.2: Greedy forwarding failure.

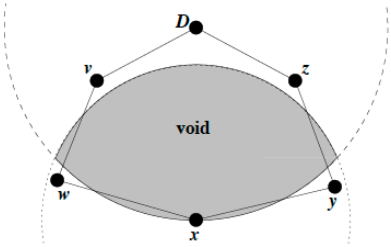

Figure 3.3: Node x's void with

3.1.0.2 The Right-Hand Rule: Perimeters

The long-known right-hand rule for traversing a graph is depicted in Figure 3.4. This rule states that when arriving at node x from node y, the next edge traversed is the next one sequentially counter-clockwise about x from edge (x,y). It is known that the right-hand rule traverses the interior of a closed polygonal region (a face) in clockwise edge order-in this case, the triangle bounded by the edges between nodes x, y, and z, in the order $(x \rightarrow y \rightarrow z \rightarrow y)$. The rule traverses an exterior region, in this case, the region outside the same triangle, in counter-clockwise edge order.

In Figure 3.3, traversing the cycle $(x \rightarrow y \rightarrow z \rightarrow D \rightarrow z \rightarrow y \rightarrow x)$ by the right-hand rule amounts to navigating around the pictured void, specifically, to nodes

closer to the destination than x (in this case, including the destination itself, D). The sequence of edges traversed by the right-hand rule is called a perimeter.
Unfortunately, the right-hand rule does not yield a traversal of the perimeter of a closed polygon on all wireless network graphs. On graphs with edges that cross, the right-hand rule may instead take a degenerate tour of edges that does not trace the boundary of a closed polygon. Such graphs with crossing edges are known as non-planar graphs, or more precisely, non-planar embeddings of graphs; for brevity, we refer to them as non-planar graphs herein. An example of a non-planar graph appears in Figure 3.5. Here, when x originates a packet to u, the right-hand rule results in the tour: $(x \to u \to z \to w \to u \to x)$. The problem is the crossing edges: (w, z) and (u, x). If (w, z) were removed from the graph, the perimeter probe from x to u would instead have taken the desired tour, $(x \to u \to z \to v \to x)$.
Authors in [50] introduce the *no-crossing heuristic*: if, during traversal of a graph by the right-hand rule, the candidate next edge crosses an edge taken earlier in the traversal, that candidate next edge is ignored, and the next edge in counter-clockwise order is taken, instead. The purpose of this heuristic is to remove crossing edges from the graph, so that the right-hand rule takes the intended tour. In the case of figure 3.5, starting from x, after taking the path $(x \to u \to z)$, the the no-crossing heuristic ignores edge (z, w), because it crosses the previously taken edge (x, u). Here, the heuristic has the desired effect: the complete clockwise outer edge tour $(x \to u \to z \to v \to x)$ is taken. The implementation of this heuristic is straightforward: each node appends its location to packets it forwards by the right-hand rule, and checks whether a candidate next edge crosses one already taken in the packet's path history using simple simultaneous equations for the two edges in question.

3.1.0.3 Planarized Graphs

While the no-crossing heuristic empirically finds the vast majority of routes (over 99.5% of the $n(n-1)$ routes among n nodes) in randomly generated networks, it is unacceptable for a routing algorithm persistently to fail to find a route to a reachable node in a static, unchanging network topology. In this section are presented alternative methods for eliminating crossing links from the network.
A graph in which no two edges cross is known as *planar*. A set of nodes with radios, where all radios have identical, circular radio range r, can be seen as a graph: each

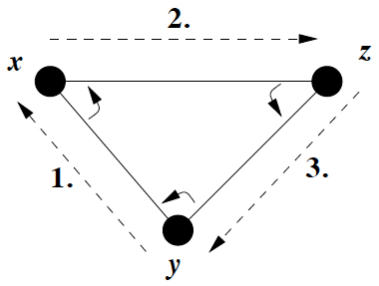

Figure 3.4: The right-hand rule

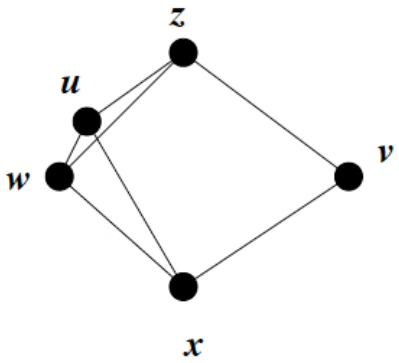

Figure 3.5: A network with crossing edges

node is a vertex, and edge (n, m) exists between nodes n and m if the distance between n and m, $d(n, m) \leq r$. Graphs whose edges are dictated by a threshold distance between vertices are termed unit graphs.
The Relative Neighborhood Graph (RNG) and Gabriel Graph (GG) are two planar graphs long-known in varied disciplines [60], [61]. An algorithm for removing edges from the graph that are not part of the RNG or GG would yield a network with no crossing links. For our application, the algorithm should be run in a distributed fashion by each node in the network, where a node needs information only about the local topology as the algorithm's input. However, for this strategy to be successful, one important property must be shown:

Removing edges from the graph to reduce it to the RNG or GG must not disconnect the graph; this would amount to partitioning the network.

Given a collection of vertices with known positions, the RNG is defined as follows:

An edge (u, v) exists between vertices u and v if the distance between them, $d(u, v)$, is less than or equal to the distance between every other vertex w, and whichever of u and v is farther from w. In equational form:

$$\forall w \neq u, v : d(u, v) \leq max\lceil d(u, v), d(v, w)\rceil \tag{3.1}$$

Figure 5 depicts the rule for constructing the RNG. The shaded region, the lune between u and v, must be empty of any witness node w for (u, v) to be included in the RNG. The boundary of the lune is the intersection of the circles about u and v of radius $d(u, v)$.
When we begin with a connected unit graph and remove edges not part of the RNG, note that we cannot disconnect the graph. (u, v) is only eliminated from the graph when there exists a w within range of both u and v. Thus, eliminating an edge requires an alternate path through a witness exist. Each connected component in an unobstructed radio network will not be disconnected by removing edges not in the RNG.

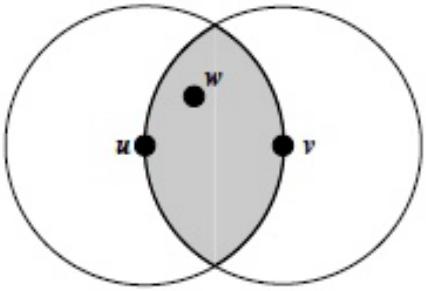

Figure 3.6: The RNG graph

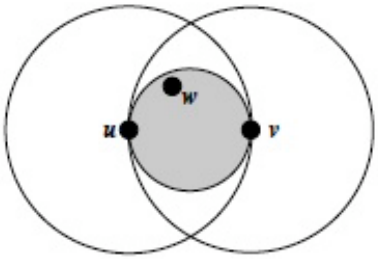

Figure 3.7: The GG graph

The GG is defined as follows:

An edge (u;v) exists between vertices u and v if no other vertex w is present within the circle whose diameter is uv. In equational form:

$$\forall w \neq u, v : d^2(u, v) \leq max\lceil d^2(u, v) + d^2(v, w)\rceil \tag{3.2}$$

Figure 3.7 depicts the GG graph membership criterion.
Eliminating edges in the GG cannot disconnect a connected unit graph, for the same reason as was the case for the RNG. Both these algorithms for rendering the graph of the radio network planar take time $O(deg^2)$ at each node, where deg is the node's degree in the full radio graph.
It has been shown in the literature [61] that the RNG is a subset of the GG. This

is consistent with the smaller shaded region searched for a witness in the GG, as compared with in the RNG. Figure 7 shows a full unit graph corresponding to 200 nodes randomly placed on a 2000-by-2000 meter region, with radio ranges of 250 meters; the GG subset of the full graph; and the RNG subset of the full graph. Note that the RNG and GG offer different densities of connectivity by eliminating different numbers of links. Many MAC layers exhibit drastically reduced efficiency as the number of mutually reachable sending stations increases [62], [63]. Moreover, while any packet a node transmits monopolizes the shared channel within its radio range, MAC protocols that address the hidden terminal problem, including 802.11 [64], MACA [65], and MACAW [66], deliberately spread contention to the full radio ranges of both sender and receiver. Under such regimes, using fewer links in routing can improve spatial diversity.

3.1.0.4 Combining Greedy and Planar Perimeters

In this section we present the full Greedy Perimeter Stateless Routing algorithm, which combines greedy forwarding on the full network graph with perimeter forwarding on the planarized network graph where greedy forwarding is not possible. Recall that all nodes maintain a neighbor table, which stores the addresses and locations of their single-hop radio neighbors. This table provides all state required for GPSR's forwarding decisions, beyond the state in the packets themselves.
The packet header fields GPSR uses in perimeter-mode forwarding are shown in Table 3.1. GPSR packet headers include a flag field indicating whether the packet is in greedy mode or perimeter mode. All data packets are marked initially at their originators as greedy mode. Packet sources also include the geographic location of the destination in packets. Only a packet's source sets the location destination field; it is left unchanged as the packet is forwarded through the network.
Upon receiving a greedy-mode packet for forwarding, a node searches its neighbor table for the neighbor geographically closest to the packet's destination. If this neighbor is closer to the destination, the node forwards the packet to that neighbor. When no neighbor is closer, the node marks the packet into perimeter mode.
GPSR forwards perimeter-mode packets using a simple planar graph traversal. In essence, when a packet enters perimeter mode at node x bound for node D, GPSR forwards it on progressively closer faces of the planar graph, each of which is crossed by the line $\overline{xD}$. A planar graph has two types of faces. Interior faces are the closed

polygonal regions bounded by the graph's edges. The exterior face is the one unbounded face outside the outer boundary of the graph. On each face, the traversal uses the right-hand rule to reach an edge that crosses line $\overline{xD}$. At that edge, the traversal moves to the adjacent face crossed by $\overline{xD}$. See Figure 8 for an example. Note that in the figure, each face traversed is pierced by xDthe first two and last faces are interior faces, while the third is the exterior face.

When a packet enters perimeter mode, GPSR records in the packet the location Lp, the site where greedy forwarding failed. This location is used at subsequent hops to determine whether the packet can be returned to greedy mode. Each time GPSR forwards a packet onto a new face, it records in Lf the point on $\overline{xD}$ shared between the previous and new faces. Note that Lf need not be located at a node; $\overline{xD}$ usually intersects edges, as in Figure 8. Finally, GPSR records $e0$, the first edge (sender and receiver addresses) a packet crosses on a new face, in the packet.

Upon receiving a perimeter-mode packet for forwarding, GPSR first compares the

Table 3.1: GPSR packet header fields used in perimeter mode forwarding.

Field	**Function**
D	Destination Location
Lp	Location Packet Entered Perimeter Mode
Lf	Point on $\overline{xV}$ Packet Entered Current Face
e0	Frst Edge Traversed on Current Face
M	Packet Mode: Greedy or Perimeter

location Lp in a perimeter-mode packet with the forwarding node's location. GPSR returns a packet to greedy mode if the distance from the forwarding node to D is less than that from Lp to D. Perimeter forwarding is only intended to recover from a local maximum; once the packet reaches a location closer than where greedy forwarding previously failed for that packet, the packet can continue greedy progress toward the destination without danger of returning to the prior local maximum.

When a packet enters perimeter mode at x, GPSR forwards it along the face intersected by the line $\overline{xD}$. x forwards the packet to the first edge counter-clockwise about x from the line $\overline{xD}$. This determines the first face over which to forward the packet. Thereafter, GPSR forwards the packet around that face using the right-hand rule. There are two cases to consider: either x and D are connected by the graph, or they are not.

When x and D are connected by the graph, traversing the face bordering x in either direction (we use the previously described right hand rule) must lead to a point y at which xD intersects the far side of the face. This is the case whether the traversed face is interior or exterior. At y, GPSR has clearly reduced the distance between the packet and its destination, in comparison with the packet's start in perimeter mode at x.

While forwarding around a face, GPSR determines whether the edge to the chosen next hop n intersects $\overline{xD}$. GPSR has the information required to make this determination, as Lp and D are recorded in the packet, and a GPSR node stores its own position and those of its neighbors. If a node borders the edge where this intersection point y lies, GPSR sets the packet's Lf to y. At this point, the packet is forwarded along the next face bordering point y that is intersected by xD. The node forwards the packet along the first edge of this next face-by the right-hand rule, the next edge counter-clockwise about itself from n. This first edge on the new face is recorded in the packet's $e0$ field.

This process repeats at successively closer faces to D. At each face, the packet progresses by the right-hand rule until reaching the edge that intersects with $\overline{xD}$ at a point y closer than the packet's Lf field to D. Finally, the face containing D is reached, and the right-hand rule leads to D along that face.

When D is not reachable (i.e., it is disconnected from the graph), two cases exist: the disconnected node lies either inside an interior face, or outside the exterior face. GPSR will forward a perimeter mode packet until the packet reaches the corresponding face. Upon reaching this interior or exterior face, the packet will tour unsuccessfully around the entirety of the face, without finding an edge intersecting $\overline{xD}$ at a point closer to D than Lf. When the packet traverses the first edge it took on this face for the second time, GPSR notices the repetition of forwarding on the edge e0 stored in the packet, and correctly drops the packet, as the destination is unreachable; the perimeter-mode graph traversal to a reachable destination never sends a packet across the same link in the same direction twice.

Note that GPSR will greedily forward a packet for potentially many hops, before the packet loops on an exterior or interior face and is recognized as undeliverable. If the majority of unreachable destinations lie beyond the boundary of a single face, undeliverable packets may concentrate at that face of the network graph. This behavior is a direct consequence of GPSR's avoidance of transitive routing protocol traffic across the many hops from a destination to a forwarding router. Other tech-

niques for scaling routing have similar effects, however: the hierarchy used to scale routing on wired networks obscures intra-domain link failures from the backbone in the interest of scaling. Thus, the inter-domain routing system will push a packet a great distance, with the potential result that the packet will be dropped inside the destination AS.

3.1.1 GPSR Trust Model

The GPSR Trust Model that is presented here is the one proposed in the AWISSENET (Ad-hoc personal area network and WIreless Sensor SEcure NETwork) project trust scheme [67] for the GPSR routing protocol. A similar Trust Model is also proposed in [68].

The primary concept of the proposed trust model is to create on each sensor a trust repository (Trust Table), which will maintain and handle trust and reputation information about each neighbouring node. The Trust Table will calculate and store the values of a number of events and parameters and by applying the proper weighting factor to each one, they contribute towards the selection of the proper forwarding node. As a conclusion, reputation and trust are two very useful tools that can be used in order to facilitate decision making in WSN networks, and thus, trust management can effectively improve network performance and help towards detection of malicious node behavior. Therefore, specific trust metrics can be measured to efficiently address specific security or routing protocol attacks or trust model security attacks.

The outcome of the trust model will be the forwarding trustworthiness of each node and will be exploited to define secure routes to the destination. This trust model will be based on both direct and indirect trust values while its functionality will be distributed, as will be described later in this section. The trust metrics (in other words the type of events) that will be monitored will be discussed below and may be different for different types of devices to achieve applicability to heterogeneous environments and allow for lifetime security trade-offs. The protocol can discover multiple paths between two nodes. This is essential for an ad hoc network to be able to tolerate attacks inducing path failures and provide robust packet delivery.

3.1.1.1 Trust Model Architecture

The proposed trust model is a decentralized trust scheme, i.e. the trust management functionality will be distributed over the network nodes. The rationale behind this choice is the following: Centralized trust requires a centralized, globally trusted node, which computes the trust of every node in the network. Apparently, a node has to select information about another node by asking the centralized node. This notion has two disadvantages:

1. By capturing the centralized node, the whole network is under capture (single point of failure)
2. In this way, the personal opinion regarding a node is suppressed (especially in cases where the types of events collected by the centralized node refer to network parameters, such as link quality, or latency).

Following the decentralized approach, each node is responsible for computing its own trust value per relation in the network, either collecting events from direct relations, or collecting trust values from other nodes in the network. It is also assumed a decentralized trust scheme, where no central authority is present and every decision is up to the individual node of the network. However, an option is considered in order to assign higher weights to reputations (indirect trust information) coming from nodes in higher layers of the architecture. The only assumption for this model is that nodes will be pre-programmed with appropriate software (depending on their role) and pre-arranged keys will be distributed for secure communication.
Both direct and indirect trust values will be used to evaluate each node's trustworthiness. The indirect (second-hand) information is particularly useful when no or limited direct interaction has been attempted. In this concept, every node can build a relation with its neighbors, based on the collection of actions (events) performed by other nodes in the neighborhood.

In Figure 3.8, a high-level description of the proposed trust model architectures presented, as it will be implemented on each network node. Each node monitors events which are then stored by the trust management component to the direct interaction trust table. To evaluate the trustworthiness of a certain node, indirect information is required and is obtained through the following sequence of actions: a reputation request is broadcasted and the received responses are delivered to the

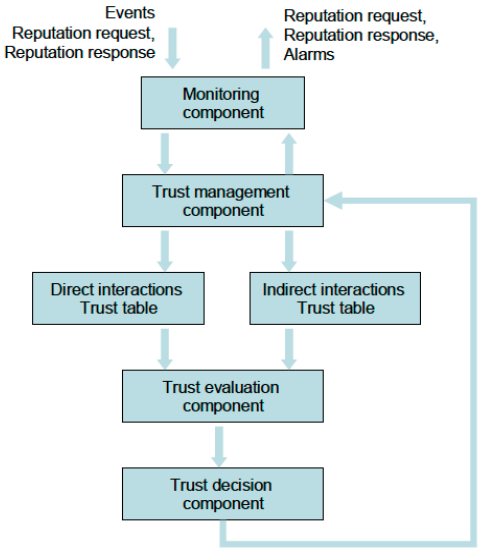

Figure 3.8: General Architecture of the Trust Model

trust management component through the monitoring component. The content of the reputation response is stored in the indirect interactions table, if a set of checks that will be detailed later on is successful. Finally, the trust evaluation component combines the direct and indirect interactions information to calculate a single trust value per neighboring node and finally forward these values to the trust decision component. This component is also triggered when the node is asked to provide its trust value for a neighbor, i.e. to form a reputation response message.

3.1.1.2 Events and Data Collection

One of the most important aspects of trust management schemes is the process of data collection. In general, for the development of a trust management system, data related to the neighboring nodes behavior is collected and then analyzed depending on how the trust management system works. The direct trust value of a neighboring node can be determined by its multi-attribute, time-varying trust value depending on a set of events. The information related to previous cooperation is assembled in a Trust Table, as shown in. Direct Trust Table. Therefore, it is essential to point out, what type of data will be more relevant for these systems,

and which are the events that can provide a useful feedback to the system, towards the proper decision.
The structure of the Trust Table that stores the trust values is shown in Direct Trust Table 3.2. Each node with k neighboring nodes will store k Trust Tables. Thus, the table size should be as small as possible (especially in densely deployed network scenarios), while keeping the most important information. As shown for each event, are stored both the number of successful and the number of failed interactions. In

Table 3.2: Direct Trust Table Structure

Reference Node	**Reference Node ID/Address**
Forwarding	number of Success/Failures
Network-ACK	number of Success/Failures
Packet precision-Integrity	number of Success/Failures
Authentication	number of Success/Failures
Cryptography-Confidentiality	number of Success/Failures
Reputation RES	number of Response/no Response
Reputation Validation	Value
Remaining Energy	Value
Network ACK History Log	1011010011010111
Number of Interactions	Value
Distance of Sink Node	Value

order to take the final forwarding decision, the trust values will be combined with factors like the distance to base station, number of hops to base station and node confidence. A subset of the above described events can be monitored and used to evaluate a node's trustworthiness by each type to allow for the trust model's adaptation to the application needs and node's features.

3.1.1.3 Direct Trust Evaluation

For each one of the first 6 events of Table 3.2, node's A Trust value regarding node B, i.e. $T_i^{A,B}$, can be calculated as follows:

$$T_i^{A,B} = \frac{a_i S_i^{A,b} - b_i F_i^{A,B}}{a_i S_i^{A,B} + b_i F_i^{A,B}} \tag{3.3}$$

where:

- $S_i^{A,B}$ is the number of successful type i events that A has measured for B
- $F_i^{A,B}$ is the number of failed type i events that A has measured for B
- a_i and b_i represent the weight/significance of a success vs. the weight/significance of a failure of type E_i events.

For event types 7 and 8 of the table 3.2, the trust value is the value already stored at the table. This value is increased or decreased based on periodic monitoring. Especially for type 8, the equation below (or a similar one, based on the simulations) will be used:

$$T_8^{A,B} = \frac{a_8 V_{now} - b_8 V_{initial}}{a_8 V_{now} + b_8 V_{initial}} \tag{3.4}$$

where V_{now} is the latest voltage value of node B and $V_{initial}$ is the initial voltage value of node B.

For the History Log a simple pattern matching technique is used which will help towards either calculating the trust value or categorizing the neighboring nodes activity.

The number of interactions is a measure of confidence. A high confidence value means that the target has passed a large number of tests that the issuer has set, or that the issuer has interacted with the target for a long time, and the node is sure that the value of the neighboring node is more certain. The algorithm of trust evaluation is more sensitive in the beginning of the interactions period (since confidence value is small, one fault should have a large impact in trust value), while as confidence value increases, the impact (either on positive or negative events) is smoother. Thus, a confidence factor is defined, like in the next equation:

$$C^{A,B} = 1 - \frac{1}{noi + a_{10}} \tag{3.5}$$

where noi indicates the number of interactions with node B and a_{10} is a factor whose value will be checked during simulation testing. This confidence factor can be proved to be useful, especially during the beginning of network operation. Moreover, in case of GPSR a proper metric that can be implemented in this trust model which is the distance of each one of the neighboring node to the sink. The closer a node to the sink, the greater the value added to the final direct trust of the node.

Finally, node's A Direct Trust value for its neighboring node B, i.e. $DT^{A,B}$ with k event types can be calculated according to the following equation:

$$DT^{A,B} = C^{A,B} * (\sum_{i=1}^{k} W_i * T_i^{A,B}) \tag{3.6}$$

where W_i is the weighting factor for each one of the k event types and $T_i^{A,B}$ is node's A trust value of event i regarding node B.

3.1.1.4 Indirect Trust/Reputation Evaluation

There are several cases where a node (e.g. node A) needs the trust opinion of its neighboring nodes (e.g. node C, D, E) regarding a specific node (node B). Examples of such cases may be the discovery of a new node appeared during a "HELLO" message or when direct trust value of node B is neutral (its value is neither large nor small). In this trust model, a node A may find the indirect trust/reputation value of a node B i.e. the $IT^{A,B}$ by combining the direct trust values (reputation values) of its neighboring nodes, as shown in the following equation:

$$IT^{A,B} = \sum_{j=1}^{n} W(DT^{A,N_j}) * DT^{N_j,B} \tag{3.7}$$

where n is the number of neighboring nodes to A, N_j are neighboring nodes to A, $DT^{N_j,B}$ is node's N_j reputation value of node B and $W(DT^{A,N_j})$ is a weighting factor reflecting node's A direct trust value of node N_j. Different weighting factors are used for each node regarding the events described above. For example, if node's C direct trust value (evaluated by node A) is large and also node C is frequently sending responses to node's A requests, then its weighting factor is large.
The reputation value B ($DT^{N_j,B}$) that the neighboring nodes propagate to the interested node are kept to the Reputation Indirect Trust Table, thus the interested node can check the correctness of their answers on next route discovery phase and modify the direct trust values of the neighbors $W(DT^{A,N_j})$ accordingly (e.g. increase the direct trust value of a node who gave a reputation that was proved correct). This is the reason of the direct trust value selection, instead of the sum of direct and indirect trust values.
The metrics that allow node A to evaluate node's B trustworthiness in this case are the node's direct trust value, which includes its responsiveness in the reputation

scheme implementation as well as the provided reputation value. Considering very important the node willingness to participate in the reputation scheme, the history of this interaction is also maintained to allow for fast detection of nodes injecting incorrect reputation values implementing either bad-mouthing attacks or attacks based on colluding nodes.

Table 3.3: Indirect Trust Table Structure

Direct Trust Value	**Value**
Reputation RES	number of Response/no Response
Reputation value of responding node	value
Reputation Correctness History Log	10110100110101

3.1.1.5 Total Trust Evaluation

The total trust evaluation node A of node B, i.e. $TT^{A,B}$ is performed by applying the following equation:

$$TT^{A,B} = W(DT^{A,B}) * DT^{A,B} + W(IT^{A,B}) * IT^{A,b} \tag{3.8}$$

where $DT^{A,B}$ is node's A trust value of node B, $W(DT^{A,B})$ is a weighting factor reflecting node's A direct trust value of node B, $IT^{A,B}$ is a node's A indirect trust value of node B and $W(IT^{A,B})$ is a weighting factor reflecting node's A indirect trust value of node B. The weights can possibly be set to zero, and be dynamically updated and adjusted by the nodes to reflect their own conditions. Every time that the cooperation among the nodes is completed, every node records and updates the trust value of its cooperation node. The trust value of a node varies with time. In our trust model, if a node cannot provide cooperation for other nodes, the other nodes will gradually decrease its trust value accordingly. Since node A can be sure only about the first-hand information that has collected, the weighting factor of the Direct Trust Value will be larger than the weighting factor of the Indirect Trust value. This remark might not be applicable in the case where a new node appears in the neighborhood, where indirect information may be the only source of information to be used for validating the neighbour's trustworthiness. This case will also be examined through computer simulations.

3.1.2 GPSR Software Implementations

In this section we describe in detail all the existing software implementations of the GPSR routing protocol. So far, there are three different software implementations : A PC/Linux Based, a TinyOS-1.x and a TinyOS-2.x [69] one. Our hardware approach (which will be described in detail in the next chapter), is implemented and verified based on these software implementations.

3.1.2.1 C-Based Implementation

In this section we describe in detail, the functionalities of GPSR implementation, the structure of software, sequence diagrams for normal functionalities and format of all GPSR protocol packets [70], [71]. This is based on [50] and is run-able on the following Linux Systems:

1. PCs/WSs running Red-hat Linux 8.0/9.0 [72] or Ubuntu 9.04 [73]
2. PC-104 running Linux + Real mote (Mica) [74]
3. HP IPAQ running familiar Linux [75]

Regarding the functions of GPSR implementation these are divided into:

1. States of Neighbors: This keeps states based on positioning information from all immediate neighbors and provides the following functions for other modules:
   - Add and delete neighbor
   - Update and look-up the state of neighbor
   - Update RNG topology and GG topology
   - Find shortest-path for greedy forward
   - Find clockwise-path for perimeter forward
2. Beaconing
   - Receive beacon packet and beacon solicit packet
   - Periodically broadcast beacon packet to neighbor nodes

- Periodically check connections to neighbor nodes

3. Greedy Forwarding
   - Receive greedy-mode data packet
   - Send the pure application data except for GPSR protocol header to upper application if the destination of packet is same as the position of local node.
   - Send the receiving whole packet to upper application if GPSR daemon runs on loosely-coupled mod.
   - Forward the packet to shortest neighbor nodes if GPSR daemon runs on tightly-coupled mod.
4. Perimeter Forwarding
   - Receive perimeter-mode data packet
   - Send the pure application data except for GPSR protocol header to upper application if the destination of packet is same as the position of local node.
   - Send the receiving whole packet to upper application if GPSR daemon runs on loosely-coupled mod.
   - Forward from on perimeter mode to on greedy mode if distance from local to destination is shorter than one from previous node to destination.
   - Drop(send to application) the pure data except for protocol header if receiving packet is the previous sent packet to neighbor.
   - Perform "face change" algorithm.
   - Forward the packet to counter-clockwise neighbor nodes.
5. Upper interface
   - Send data to $GPSR_{API}$ who is a communication agent for applications.
   - Trigger $greedy-mode$ or $perimeter-mode$ forward if data from $GPSR_{API}$ are received.
   - Create a thread per a receiving data from $GPSR_{API}$.
6. Lower interface

- Unicast data packet to neighbor node or broadcast beacon to all neighbors via communication socket.
- Trigger $greedy-mode$ or $perimeter-mode$ reception if data from neighbor nodes are received.
- Create a thread per a receiving data from neighbor nodes.

GPSR is implemented as Daemon process and it consists of two modules:

1. GPSR daemon that performs original GPSR functions and
2. Application Programming Interface (API) library that provides the access to GPSR for various applications.

The characteristics of GPSR implementation are as follows:

1. **Event-driven:** when GPSR protocol packets from neighbor nodes or application messages from applications are received, polling is not used because it can unnecessarily consume power resources during idle periods and also inadequate for real-time attributes that wireless applications can own.
2. **Multithread:** a thread per an event(reception of packet, reception of application data, timeout) is created.
3. **Multiplexing:** GPSR daemon can establish multiple channels for multiple applications. It is intended for multiple applications concurrently use GPSR daemon. To discriminate a destination application when a GPSR data packet is received, GPSR daemon uses multiplexing table and app_{port} within GPSR packet header.
4. **Use of UDP socket:** UDP socket is used as communication with applications.
5. **Support of Ethernet, 802.11.b, mote radio**: they can be used for communication with neighbor nodes.

3.1.2.2 TinyOS Based Implementations

In [76] there is a detailed description of the TinyOS-based GPSR implementation, as well as a complete specification for programmers who intend to make TinyOS-based applications.
Regarding the version of TinyOS used, there are so far two different implementations, one for TinyOS-1.x and one for TinyOS-2.x, which have the same functionality. The motes that this software implementations are tested on are the Crossbow's [74] Micaz and Iris [77] ones for both TinyOS versions
This full-fledged nesC [78] implementation includes the GG and the RNG planarization algorithms (chosen via a configuration parameter), as well as greedy and perimeter mode packet forwarding.
Here are described the functionalities of the TinyOS GPSR Implementation, as these presented in [76]. According to the authors, these are divided in the following categories:

1. **LLC (Logical Link Control):**
    - It fragments a long GPSR packet into multiple TOS messages and reassembly multiple TOS messages to a long GPSR packet.
    - It can compress sending packets and decompress receiving packets if they are compressed.
    - Reliable packet delivery can be performed through sequencing and hop-by-hop acknowledgment.
    - It provides a link-probing function that performs probing-test on a specific link and eliminates highly noisy or asymmetric links according to result of probing-test.
2. **Neighbor List.** This keeps state such as positioning information of immediate neighbors and performs the followings:
    - Adds neighbors to neighbor list and delete neighbors from neighbor list
    - Updates and looks up state of neighbors
    - Finds next-hop for greedy forward
    - Finds clockwise-path next-hop for perimeter forward

- Finds counter-clockwise-path next-hop for some purposes.

3. **Beaconing:**
   - Periodically sends beacon on the radio
   - Receives beacons from neighbors and processes them as follows: If sender of beacon is new on neighbor list, information about sender is registered to neighbor list, link probing function is triggered to measure link-quality, and then planarization task is triggered to reflect topology change if result of link-probing is well. If sender is already registered on neighbor list, some information about sender is update to neighbor list. Optionally, events such as addition/deletion/update of neighbor can be sent to applications
   - Periodically checks connectivity to all nodes on neighbor list: If connectivity for a neighbor is failed, link probing is triggered. If result of link probing is bad, information for the neighbor is deleted from neighbor list and then planarization task is triggered to reflect topology change.
4. **Planarization:**
   - Whenever topology is changed, planarization is triggered: Planarization generates Gabriel Graph or Relative Neighborhood Graph from full graph. The planarized graph is determined when applications initialize GPSR component. Gabriel Graph planarization is used as default.
   - According to result of planarization, some links on the neighbor list are marked as routable to be used by perimeter forward and other links is marked as non-routable not to be used by perimeter forward. Optionally, before a link is marked as non-routable, Mutual Witness query can be sent to corresponding neighbor.
5. **Mutual Witness Protocol**
6. **Packet reception**
7. **GPSR Forward (Greedy and Perimeter):**
   - It firstly tries to find a greedy neighbor to destination

- If a greedy neighbor is found: If it is performed on intermediate node, distance from greedy neighbor to destination and one from Lf to destination is compared. If first one is shorter than second one, forward mode for packet is change to greedy mode.
- If a greedy neighbor is not found, forward mode for packet is changed to perimeter mode. It tries to select a clockwise neighbor by right-handed rule and face-change rule. If selected clockwise neighbor is same as Lf in the packet, the packet is dropped (transfer it to application)
- After a neighbor to which packet is sent is found, packet is transferred to LLC to be sent to a next hop.

8. **Broadcast Forward and UART communication**

Because GPSR is implemented on TinyOS programming platform, it has the following characteristics:

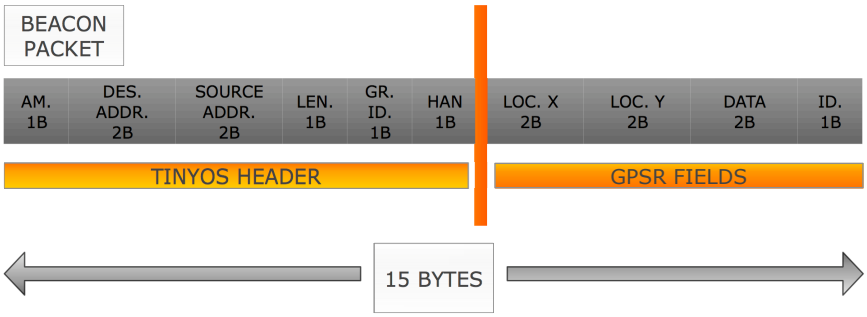

Figure 3.9: GPSR Beacon Packet Format

1. **Component Based:** GPSR software is a component program using TinyOS system components. GPSR program consists of three components: GPSRForwarder, GPSRRouter, and LLC component.

2. **Event Driven:** when GPSR data packets from neighbor or from applications are received, corresponding event handlers or command handlers are invoked. Also, event handlers are invoked by time-out events.

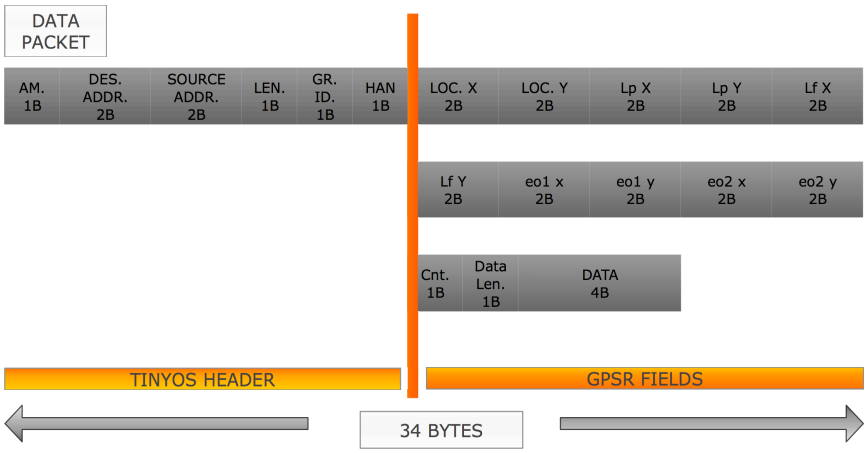

Figure 3.10: GPSR Data Packet Format

3. **Concurrency Intensive:** Because jobs like forwarding a GPSR data packet spend much time for processing, event handler or command handler is inadequate for long -term jobs. Hence, these event/command handlers fork long-term tasks and are promptly exited. According to this action, multiple tasks to forward a GPSR data packet can exist by a sequence of events or commands.

At this point we present the format of the Data and Beacon Packets of the GPSR Routing Protocol. The same packet format will be used in our Hardware Implementation of the Protocol. The size of the Beacon packets is 15 whereas the size of the GPSR Data packets (Greedy and Perimeter) is 34 bytes long. In figures 3.9 and 3.10 the format of the packets is presented whereas in Table 3.4 are described the fields of the aforementioned packets.

3.1.2.3 GPSR Secure Routing Module

A software implementation of the GPSR protocol together with the Trust model is proposed in the AWISSENET's project. The software implementation consists of the proposed nesC implementation for Tinyos 2.x of the GPSR together with the implementation of the previously described Trust Metrics for Direct, Indirect

Table 3.4: GPSR Packets Fields

Field	Beacon	Data
AM	Active Message	Active Message
DES. ADDR.	Destination Address (0xFF)	Destination Address
SOURCE ADDR.	Link Source Address	Link Source Address
LEN.	Message Length	Message Length
GR. ID	Group ID	Group ID
HAN	Handler Type	Handler Type
LOC. X	Location X of node	Location X of node
LOC. Y	Location Y of node	Location Y of node
DATA	Transmitted Data	-
ID	Node ID	-
Lp X	-	Enter Perimeter Location X
Lp Y	-	Enter Perimeter Location Y
Lf X	-	Enter Face Location X
Lf Y	-	Enter Face Location Y
eo1 X	-	First Edge Start X
eo1 Y	-	First Edge Start Y
eo2 X	-	First Edge End X
eo2 Y	-	First Edge End Y
Cnt.	-	Control Flag
Data Len.	-	Data Lenght
DATA .	-	Transmitted Data

and Total Trust. This implementation though, is still under development, so we cannot provide further details at the moment.

3.2 Xmesh Protocol Stack

Multi-hop or ad hoc, wireless networks use two or more wireless hops to convey information from a source to a destination. There are many Multi-hop Routing protocols for WSNs, which are characterized by the absence of a single multihop stack, and also by the fact that they are application-dependent. Each of these protocols adopts a different approach for routing, for energy management and for the overall latency management. TinyOS which is the most widely used OS for WSN nodes, allows users to wire-in different protocols with minimal effort. A Multihop

protocol can be executed on each Mote, while each Mote can serve both as a data source and as a router.
There are three different routing protocols supported by TinyOS 1.x which differ in terms of both the actual routing algorithm and the services they provide. In particular, *Route* seeks to minimizes the number of hops that each packet traverses while *MINTRoute* and *ReliableRoute (XMesh)* route packets based on link-quality estimates that seek to maximize the probability of a packet being delivered ([79], [80], [81], [82], [83], [84], [85]). The most widely used such routing scheme is the XMesh due mainly to its performance when applied in real-world WSNs.

3.2.1 Protocol Implementation Framework

Figure 3.11 captures the high level interactions of all the components of the XMesh routing protocol. Each node maintains estimates of inbound (reception) link quality. Routing is based on outbound (transmission) link, so this information needs to be propagated back to the neighbors. The core component is the neighbor table which contains status and routing entries for neighbors; its fields include MAC address, routing cost, parent address, child flag, reception (inbound) link quality, send (outbound) link quality, and link estimator data structures.
Below the routing layer, all packets on the channel are snooped by the estimator,

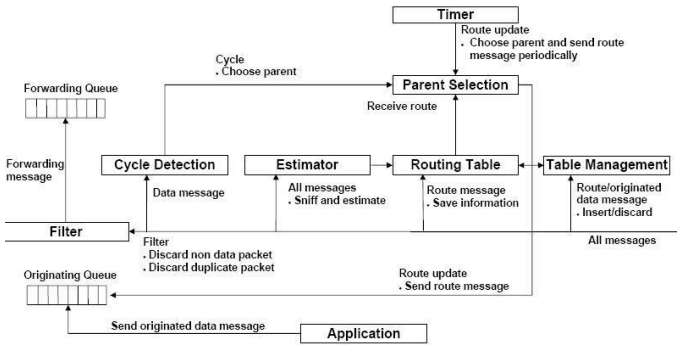

Figure 3.11: ReliableRoute Components

with insertions controlled by the neighbor table manager. Parent selection is run periodically to identify one of the neighbors for routing; it may also broadcast (locally)

a route message. The route messages include parent address, estimated routing cost to the sink, and a list of reception link estimations of neighbors. When a route message is received from a node that is resident in the neighbor table, the corresponding entry is updated. Otherwise, the neighbor table manager decides whether to insert the node or drop the update. Data packets originating from the node, i.e., outputs of local sensor processing, are queued for sending with the parent as the destination. Incoming data packets are selectively forwarded through the forwarding queue with the current parent as destination address. The corresponding neighbor table entry is flagged as a child to avoid cycles in parent selection. Duplicate forwarding packets are eliminated. When cycles are detected on forwarding packets, parent selection is triggered with the current parent demoted to break the cycle.

3.2.2 Xmesh Underlying Issues

Parent Selection: Many distance-vector based algorithms can be implemented in this framework, using different cost metrics to guide routing. The cost of a node is an abstract measure of distance; it may be number of hops, expected number of transmissions, or some other estimate of energy required to reach the sink. When scheduled to run, the routing algorithm accesses the neighbor table and extracts a set of potential parents. A neighbor is selected as a potential parent only if its cost is less than the current cost of the node. A node may switch to a new parent if one is sufficiently smaller in cost by some margin than the current parent. It may also switch to a new parent if the link quality to the current parent drops below some threshold, if the sink is unreachable through the current parent, or if a cycle is detected. When connectivity to the current parent worsens, its link estimation will automatically degrade over time, allowing the selection of a new parent.If connectivity to the current parent is lost and no potential parents are available, the node declares it to have no parent, disjoints from the tree, and sets its routing cost to infinity.

Rate of Parent Change: Regardless of the routing algorithm, routes can be changed whenever the parent selection algorithm is scheduled to run. For fast adaptation, it is tempting to schedule the parent selection component to evaluate new routes for every route update received from neighboring nodes. However, a domino effect of route changes is likely to be triggered across the entire network, especially when routing costs are very sensitive. To achieve a stable topology, routes are eval-

uated on a periodic basis, rather than upon receiving a route update, except when a cycle is detected.

Packet Snooping: Given that the wireless network is a broadcast medium, a lot of information can be extracted by snooping. Link estimation is one example. At the routing level, since each node is a router, snooping on forwarding packets allows a node to learn about all its children, which is useful to prevent cycle formation. Furthermore, snooping on a neighboring nodes messages is a quick way to learn about its parent, which decreases the chance of stale information causing a direct two-hop cycle. The same technique can also be used to prune children quickly in the case of a network partition. When a node with an unreachable route receives a forwarding message from its child, it will NACK by forwarding the childs message with a NO ROUTE address. All neighboring nodes, including its children, snooping on this packet can quickly learn about an unreachable route. In fact, this naturally provides feedback deep down into the tree, in effect solving the counting-to-infinity problem.

Cycles: For many-to-one routing over relatively stationary sensor networks, it is better to use simple mechanisms to mostly avoid loop formation and to break cycles when they are detected, rather than to employ weight protocols with inter-nodal coordination. By monitoring forwarding traffic and snooping on the parent address in each neighbors messages, neighboring child nodes can be identified and will not be considered as potential parents. This information should be maintained for nodes in the neighbor table. Route invalidation when a node becomes disjoint from a tree and tree pruning by NACKing children traffic are used to alleviate stale information, which leads to cycles. With these simple mechanisms, cycles may potentially occur and must be detected. Since each node is a router and a data source, cycles can be detected quickly when a node in a loop originates a packet and sees it returning. This mechanism works as long as queue management policy avoids letting forwarding traffic suppress originated traffic. (Otherwise, packets may get stuck in a loop in the middle of a route without detection.) This level of fairness is an appropriate policy in any case. Once a cycle is detected, discarding the parent by choosing a new one or becoming disjoint from the tree will break it. **Duplicate Packet Elimination:** Duplicate packets can be created upon retransmission when the ACK is lost. Without duplicate packet elimination, these will be forwarded, possibly causing more retransmissions and more contention, plus they waste energy. To avoid duplicate packets, the routing layer at the originating node appends the

sender ID and an originating sequence number in the routing header. To suppress forwarding duplicate packets, each parent retains the most recent originator ID and originating sequence number in child entries in the neighbor table. This approach relies on in-order packet delivery during retransmission and assumes that the neighbor table is able to track children. Alternatively a recent originator cache could be employed.

Queue Management: Nodes high in the tree forward many more messages than they originate. Care must be taken to ensure that forwarding messages do not entirely dominate the transmission queue, since it would prevent the node from originating data and undermine cycle detection. The forwarding and originating messages are separated into two queues so that upstream bandwidth is allocated according to a fair sharing policy. The policy that is used is very simple. With the assumption that originating data rate is low compare to that of forwarding messages, priority is given to originating traffic. For data collection it is possible to estimate the ratio of forwarding to originating packets by counting the descendants of each parent.

Relationship to Link Estimation: Link estimation and routing are not entirely independent. Link failure detection based on fixed number of consecutive transmission failures can be ineffective over semi-lossy links. An estimation of the link quality yields a much better judgement of link failure. With bi-directional link estimations, routing over asymmetric links can be avoided. The stability and agility of link estimation can directly affect the stability of the routes and the rate of route adaptation, especially when the estimations are combined to form a distance metric describing a path. Therefore, the final tuning of the link estimator must be done while observing its effect on routing performance.

Routing Cost Metrics: With links of varying quality, a longer path with fewer retransmissions may be better than a shorter path with many retransmissions. An alternative approach is to use the expected number of transmissions along the path as the cost metric for routing rather than traditional cost metrics for distance routing. That is, the best path is the one that minimizes the total number of transmissions (including retransmissions) in delivering a packet over potentially multiple hops to the destination. This approach is called the Minimum Transmission (MT) metric. In considering the expected number of transmissions of a link, it is important to determine link quality for both directions since losing an acknowledgement would also trigger a useless retransmission. For each link the MT cost is estimated by the

product

$$\frac{1}{linkquality_{forward}} \times \frac{1}{linkquality_{backward}} \tag{3.9}$$

which is also proposed in [86]. The distance-vector algorithm computes overall cost of a path in the same manner as hop count with these weighted hops. In addition to optimizing for something closer to the true cost, MT eliminates the need for predetermined link thresholds. However, the stability of MT routing is potentially an issue, since the MT metric utilizes link estimations in a non-linear fashion. Thus, for MT a substantial noise margin should be used in parent selection to enhance stability.

Chapter 4

Reconfigurable (CPLD) Nodes Implementation

In this chapter we describe in detail the implementation and the hardware architecture for the different reconfigurable nodes we have designed. First we describe in detail the CPLD-approach in designing reconfigurable WSN nodes. Two different protocols were utilized: the GPSR and XMesh protocol stack respectively. One part of every of these protocols was designed and implemented on a CPLD device, which communicates with a common WSN mote through a custom communication protocol.

4.1 GPSR Implementation

An overall block diagram of the CPLD-Based WSN node implementation is presented in figure 4.1. In this subsection we will provide all the implementation details of this new approach to the design of reconfigurable WSN nodes with the use of CPLDs.

The device utilized in our pioneering design is one from the Xilinx CoolRunner-II family. The CoolRunner-II CPLD family [89] utilizes Xilinx second-generation RealDigital technology so as to provide high performance, advanced features and low power consumption, all at a very low price. Featuring a 100% digital core, up to 323 MHz performance and ultra-low stand-by current, CoolRunner-II CPLDs offer a wide range of densities, plus abundant I/O, the flexibility to move from one density to another in the same package and the lowest cost per I/O pin in the industry.

The specific prototyping CPLD board utilized is the Digilent X-Board [90], which is a complete circuit development platform for Xilinx CoolRunner-II CPLD. It provides all essential support circuits for the CoolRunner-II including an on-board USB2 port which provides a data port for CPLD configuration as well as for user data transfers. This board includes a very low-cost 256 macro cell CoolRunner-II CPLD device (XC2C256) in a TQ-144 package while more than 75 CPLD signals

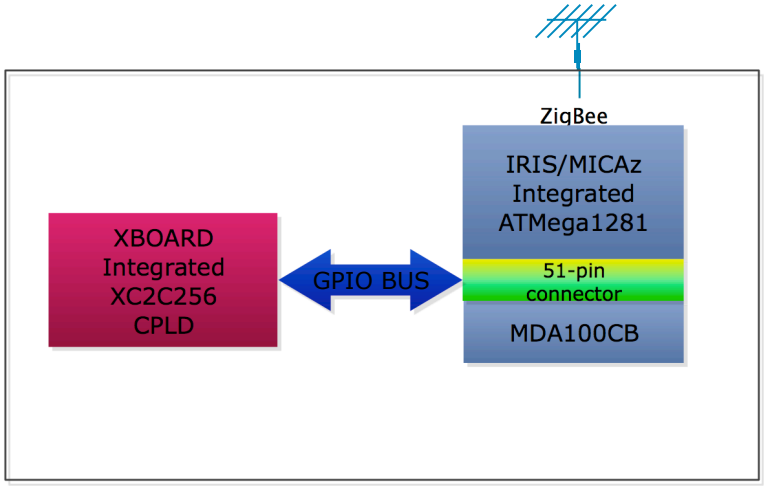

Figure 4.1: CPLD-Based Node Scheme

are routed to an expansion connector so our designs can be easily extended. Figure 4.2 presents the block diagram of Digilent X-Board.

4.1.1 Euclidean Distance Architecture

One of the main process that is involved in the GPSR routing protocol is the Euclidean Distance one. The Euclidean distance or Euclidean metric is the "ordinary" distance between two points that one would measure with a ruler, which can be proven by repeated application of the Pythagorean theorem. By using this formula as distance, Euclidean space becomes a metric space.
In our case, the Euclidean distance is used to calculate the distance between two nodes. Since every nodes is aware of the positions of all its neighbors, can easily calculate the distance between its position and every other node inside its neighbor table, in order to decide where to send a data packet.
This metric is also used in perimeter forwarding, when a node needs to decide which perimeter to follow in order for a packet to reach its destination.
The previously described metric is calculated from the following formula:

$$D(A,B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.1}$$

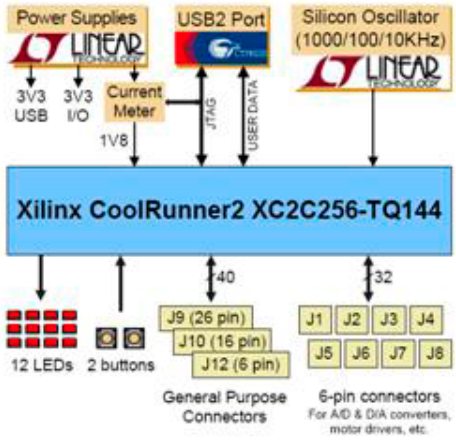

Figure 4.2: Digilent X-Board Block Diagram

where $A = (x_1, y_1)$ and $B = (x_2, y_2)$ are two dimensional points.

This metric function has been chosen in order to be implemented on the CPLD, since after some experimentation it was found that this metric is used very often by the protocol in order to calculate the necessary distances among the nodes. Besides that, we have tried also to implement on the CPLD device, other parts of the GPSR protocol, but non of them fitted inside it. The Euclidean Distance Metric is the largest part of the GPSR that fits in the chosen device.

The proposed CPLD architecture overall block diagram is presented in figure 4.3

At this point it should be mentioned that, since resources of a CPLD are very limited, the square root function was not implemented on the CPLD. On the contrary, we have chosen to implement only a part of the Euclidean Distance and especially the one that is described by the following formula:

$$(x_a - x_b)^2 \tag{4.2}$$

The rest of the Euclidean Distance function is implemented in software on the mote as it will be described in the next chapter.

As figure 4.3 depicts, the calculation module takes as inputs 4 8-bit values, which represent 2 co-ordinates. To be more specific, the first two 8-bit inputs represent the high-8 and low-8 bits respectively of x_a co-ordinate of a node's location and the second two 8-bit inputs represent, in the same manner the high-8 and low-8 of x_b

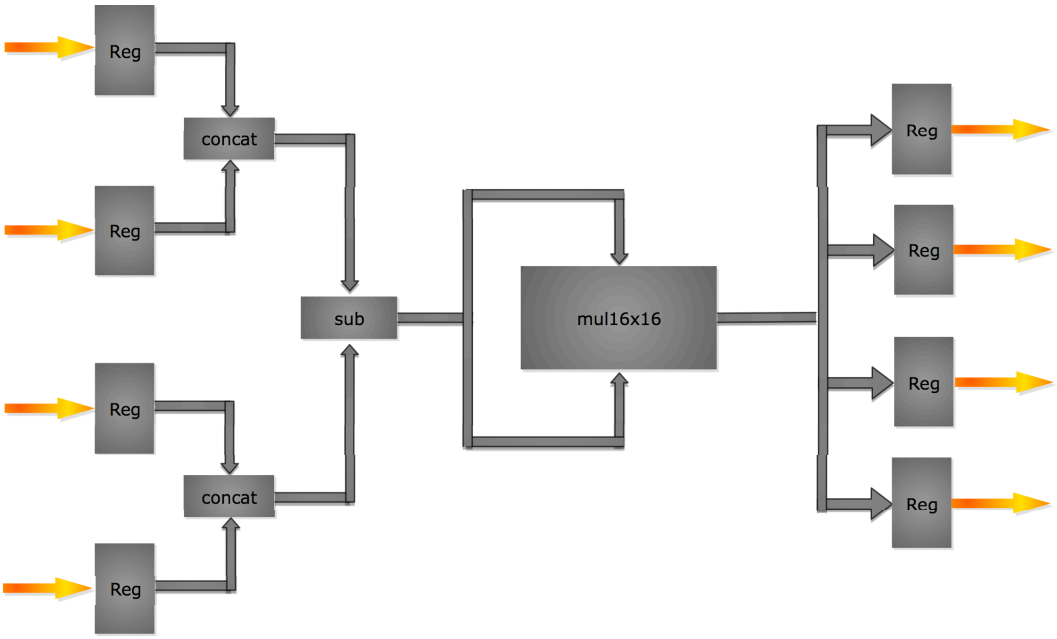

Figure 4.3: CPLD Calculation Block Diagram

coordinate of another node's location.
All these values are stored into 4 8-bit registers (the only memory element that can be used on such reconfigurable devices), since the CPLD has only 8-bit input, as we will describe in the next chapter, and thus all the input values should be stored into memory elements for later use.

Then the high and low bits of a and b co-ordinates are concatenated into 16-bit values respectively and after that b is subtracted from a. The subtracter module was implemented with the use of the VHDL libraries. At this point it should be mentioned that these values are positive unsigned integer numbers, since in a real-word environment a node can be located only in the first quartile of a Cartesian coordinate system. The comparison between these coordinates takes place in software, so there is no need to check whether there is an overflow in the result of the subtraction. Also, the range of each value can be only between [0...255], as this is defined by the GPSR protocol software.
Then the 16-bit subtraction result is squared with the use of a 16x16 multiplier in order to calculate the distance of these two coordinates. The 32-bit multiplication

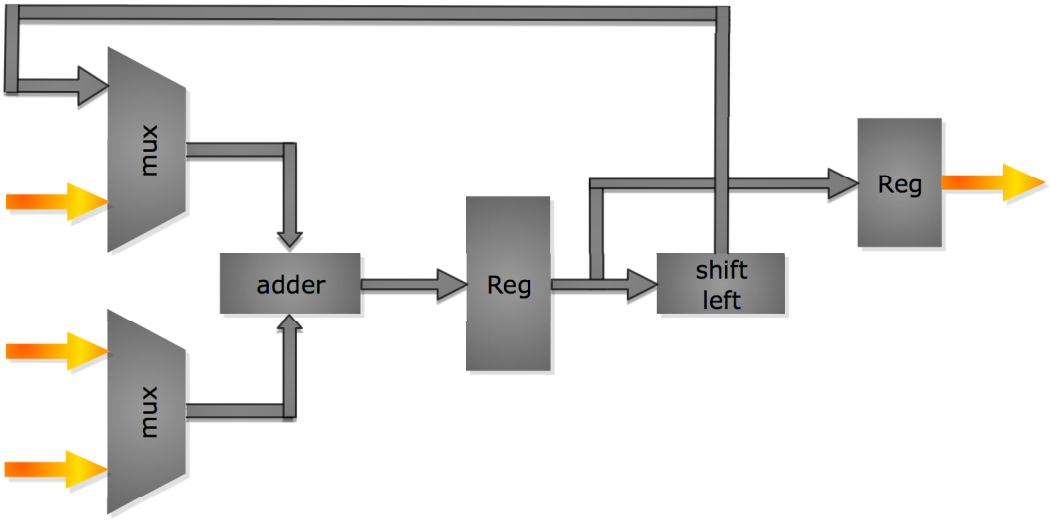

Figure 4.4: CPLD Multiplication Block Diagram

result is stored into 4 8-bit registers, so as to be transmitted back to the mote.

The 16x16 multiplier architecture block diagram is presented in figure 4.4. For the

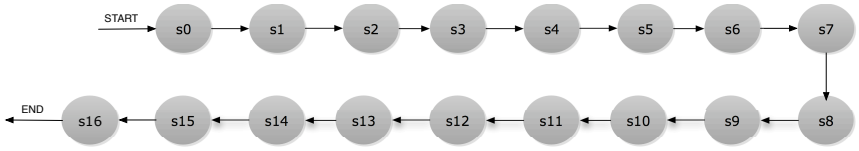

Figure 4.5: CPLD Multiplication Controller

implementation of this design, two multiplexers 2-1 of 16 bits each are used. The first multiplexer has as inputs the temporary result of the multiplication and zero. The control signal of the multiplexer chooses the zero input when the algorithm is at its first stage and the temporary result in all the other cycles. The second multiplexer has as inputs the first multiplier and a zero input. It has as a control signal every bit of the second multiplier, always according to the stage that the algorithm is in. Based on the multiplication algorithm, the process is completed in 17 cycles, one for each bit of the second multiplier plus an additional cycle that is needed in order to check and take the result from the last output register. After

the selection of the proper values, the outputs of the two multiplexers are added together with the use of a 32-bit adder; the output of the adder is stored in a 32-bit register. This register is necessary, in order to store the temporary result, since it will be later-on used for the next step of the algorithm. Then this result is shifted left by one position and goes back to the first multiplexer.
This process is repeated 16 times; during the 17th cycle the final result is stored to the 32-bit output register. For the control of this multiplier, a simple finite state machine (FSM) is designed. Each cycle of the multiplication process corresponds to a state of the designed FSM (figure 4.5).
Figure 4.6 presents the FSM that was designed in order to control the whole calculation process. It consists of 3 different cycles: During the first cycle all the inputs are stored into the registers and then the calculation process is ready to start. As soon as the calculation process is completed, a signal is triggered in order to notify the upper levels that the final result is ready so as for the output to the mote to begin.

The tool used to implement our design was the Xilinx ISE 10.1 [92], while its

Figure 4.6: CPLD Calculation Process Controller

embedded simulator was used in order to verify the correct operation of our architecture via the process of "Behavioral Simulation". Next, we had to carry out "Post Fit Simulation" and, for this purpose, we preferred Modesim SE 6.3f [93]. The CPLD was programmed using the embedded Xilinx ISE tool.

4.1.2 CPLD - Mote Interconnection

In order to implement a complete real-world WSN node, we connected the X-Board to the Crossbow MDA100 sensor and data acquisition boards [94] which include a precision thermistor, a light sensor/ photocell and provide a general prototyping area.

We have also used a USB PC Interface Board, the Crossbow MIB520 Gateway [77] which provides a USB Interface for data communication and allow the developer to seamlessly program the sensor boards.
This board has an in-system processor (ISP) which is an Atmega16L which is used to program the Motes. Code is downloaded to the ISP through the USB port and then the ISP programs the Motes processor. The motes used are the Xbow's MICAz and IRIS motes [77], which are probably the most widely used ones. MICAz motes uses the TI CC2420, IEEE 802.15.4 compliant ZigBee ready radio frequency transceiver which is integrated with an ATmega 128L micro-controller, while IRIS motes use an Atmel RF230, IEEE 802.15.4 compliant, ZigBee ready radio frequency transceiver which is integrated with an Atmega1281 micro-controller. These IRIS enhancements provide up to three times improved radio range and twice the program memory over the previous generation Motes. The block diagrams of IRIS and MICAz motes are presented in figures 4.7 and 4.8 respectively.
The development of the GPSR Routing protocol, as well as the development of cus-

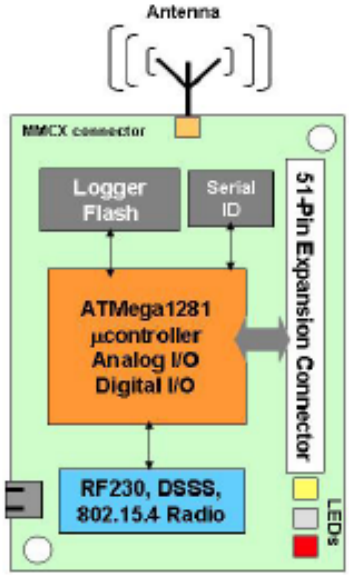

Figure 4.7: IRIS Block Diagram

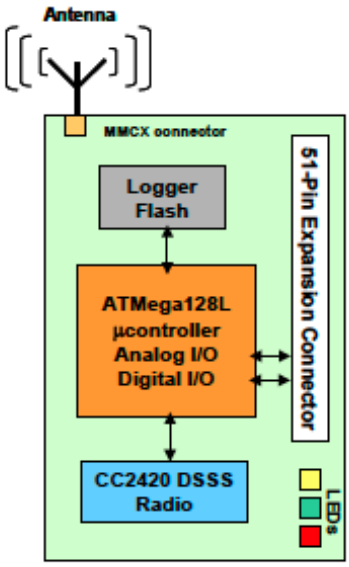

Figure 4.8: MICAz Block Diagram

tom sensor applications in enabled with the use of the TinyOS 2.1 operating system [69]. For the programming of the nodes we have utilized the nesC (network em-

bedded system C) programming language [78], which is a component-based, event-driven programming language used to build applications for the TinyOS platform. On the contrary, TinyOS is an operating environment designed to run on embedded devices used in WSNs. NesC s built as an extension to the C programming language with components "wired" together to run applications on TinyOS.
A TinyOS 2.1 implementation of the GPSR protocol, runnable on the IRIS and MICAz motes, exists and was described in a previous chapter. We have programmed all our motes with this application: The motes that are not connected with a CPLD are programmed with the given one application, whereas the other motes, which are a part of our new platform, are programmed with the a modified version of this application, in order to support the I/O process between the mote and the CPLD device.

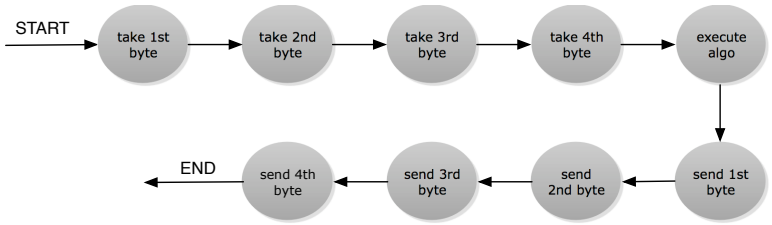

Figure 4.9: Communication Protocol Scheme

4.1.3 CPLD - Mote Intercommunication

This I/O issue was confronted with the following way: Regarding the CPLD connection, the JTAG ports were chosen for data transfers between the motes and the CPLD. For the mote connection, only 24 pins out of the 102 of the prototyping area are actually available since the remaining pins are either open or dedicated to a specific operation of the main micro-controller of the mote. Based on a traffic profiling of several applications, and since it was necessary for this connection to be used in many applications except for the GPSR one, we decided to use 8 of those pins as an input to the mote, 8 for the output traffic and the remaining for several input/output control signals 4.1.

In order to efficiently and correctly exchange data between the CPLD and the Mote,

Table 4.1: Custom Cable Pins

MDA100CB Pin	**Mode**	**CPLD Pin**	**Mode**
F2	DATA (out)	p117	DATA (in)
F3	DATA (out)	p136	DATA (in)
F4	DATA (out)	p134	DATA (in)
F5	DATA (out)	p132	DATA (in)
F6	DATA (out)	p57	DATA (in)
F7	DATA (out)	p59	DATA (in)
F8	DATA (out)	p119	DATA (in)
F13	DATA (out)	p45	DATA (in)
C10	DATA (in)	p129	DATA (out)
C11	DATA (in)	p126	DATA (out)
C12	DATA (in)	p124	DATA (out)
C13	DATA (in)	p120	DATA (out)
D10	DATA (in)	p118	DATA (out)
D11	DATA (in)	p116	DATA (out)
D12	DATA (in)	p114	DATA (out)
D13	DATA (in)	p112	DATA (out)
E2	RESET (out)	p106	RESET (in)
E3	TOGGLE (in)	p137	TOGGLE (out)
E4	OFFSET (out)	p135	OFFSET (in)

a simple toggle synchronization protocol was also implemented both in software (on the motes) and in hardware (on the CPLD). As described in the algorithm's architecture, the input to and the output of the CPLD should be 32-bit long. As a result, a number of the Mote's and CPLD's pins should be used 6 times for a single Mote-to-CPLD transfer. The protocol we have designed works as follows: firstly, the Mote sends the first 8 bits to the CPLD. This datum is stored in a register. Then the next 24 bits are accordingly sent. When the CPLD receives the first block of data, a toggle bit transits from its current state to the other, so as to ensure the data freshness and that the datum is received correctly from the CPLD. Additionally, a block-offset bit is used, to indicate which block is transferred each time. So, when the first 8 out of 32 bits are sent to the CPLD, the value of the block offset is 0, whereas the value of the block-offset bit is 1 when the byte is sent to the CPLD. After the successful reception of the first block of data, the CPLD sends an inversed toggle bit to the Mote, so as to trigger the sending of the second block of data. The

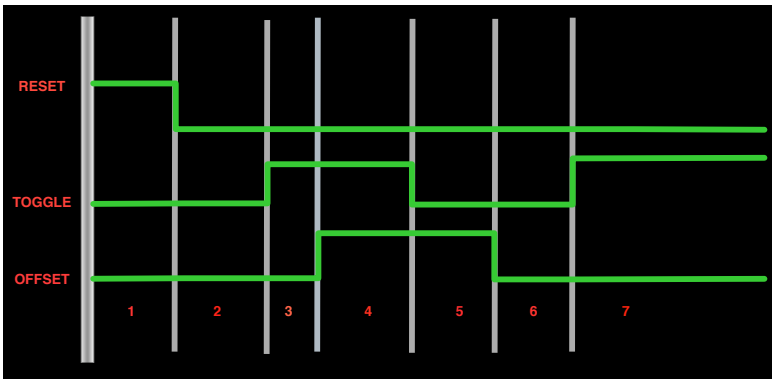

Figure 4.10: Timing Diagram of Synchronization Protocol

same procedure is followed, for all bytes, which are stored to 4 input registers in the CPLD respectively. At this point the CPLD is ready to start the processing of the received data. After the completion of the algorithm execution, the CPLD starts to send the data back to the Mote. The protocol used for sending the data to the Mote is almost the same with the one used for receiving data from the Mote: every block, is initially stored in an output register in the CPLD. Initially, a toggle bit is sent to the Mote; then the Mote replies, and upon the reception of the answer, the CPLD sends the first block with the correct value of the block offset. At this point it should be mentioned that the output of our hardware module is 32-bit long. So, every sub-block is 8-bit long, and consequently 4 blocks are sent to the Mote. Upon correctly receiving the first block of data, the Mote notifies the CPLD about this fact, and the later starts sending the second block of data to the Mote. This procedure is repeated in the same way for all the bytes. Once the transfer is finished, the Mote notifies the CPLD, and the Mote transmits the packet to the other nodes in the network.

The implementation of the communication protocol on the CPLD is presented in figure 4.9, whereas, for better explanation, a timming diagram of this protocol is presented in 4.10.Regarding the actual system integration, the Motes and the CPLD were programmed and connected together via a custom made cable. The prototyping system is presented in figures 4.11 and 4.12. The marked area is the CPLD chip, whereas the rest is the prototyping area of the Xboard development board.

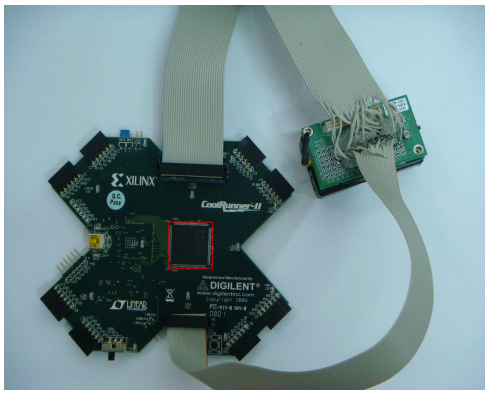

Figure 4.11: Wireless Platform Top View

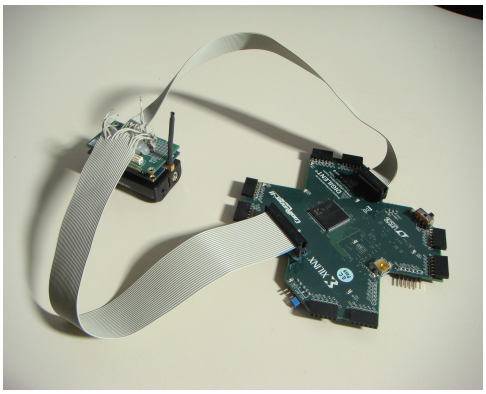

Figure 4.12: Wireless Platform

4.2 XMesh Implementation

4.2.1 Cost Functions Architecture

In order to form and maintain a Mesh network, the following two parallel processes are involved: **Link Estimation and Parent Selection**. These processes estimate the following metrics:
Receive Estimate (RE), Send Estimate (SE), Link Cost (LC), Neighbor's Cost (NC) and Overall Cost (OC). These metrics are calculated from the following equations and are described in [83]:

$$Est = 255 \times received/(received + missed) \tag{4.3}$$

$$RE = (1 - alpha) \times RE + alpha \times Est \tag{4.4}$$

$$LC = (1 \ll 18)/(SE \times RE) \tag{4.5}$$

$$OC = LC + NC \tag{4.6}$$

From the previous cost metric functions, the multiplication was chosen to be implemented on the reconfigurable device, in order to accelerate the performance of the routing protocol and to reduce the energy and the maximum power consumption. At this point, it should be mentioned that implementing all the cost metric functions on the CPLD, even though it decreases the power consumption, it also increases the whole execution time of the system, since after experimentation, it was found that the implementation of a divider in a CPLD is very slow; obviously if a larger CPLD is utilized, in which a high-speed divider can be implemented, the complete cost

function will be mapped to the CPLD resulting in greater power savings and performance improvements. Besides that, the reconfigurable multiplication unit is utilized in other tasks of the protocol as well, since there are a number of related metrics that are calculated using only multiplication. In figure 4.5 is demonstrated the block diagram of the multiplier that was implemented on the CPLD. The multiplier used is the same as the one implemented for the Euclidean Distance Architecture that is involved in the GPSR routing protocol. The only difference is that this multiplier is used for 8-bit inputs and thus the cycles that it consumes are limited to 9 (8 for the multiplication and one for the output).

4.2.2 Node Implementation

At this point it should be stressed out that for the implementation of the Xmesh Cost function that was described in the previous section the same process is involved. The same communication protocol was used, with the only difference that this process sends 2 bytes to the CPLD and receives 2 bytes respectively. Besides that, the operating system is different; since the XMesh routing protocol is no more under development, it is available only on the TinyOS 1.x, under the license of Crossbow. According to the author's knowledge, this protocol was not ported to the TinyOS 2.x platform and consequently our development was constrained only on the CPLD. Furthermore, the GPSR protocol is more efficient in terms of developing, since it's code is freely distributed and no permission is needed in order to expand it.

Chapter 5

Reconfigurable (FPGA) Nodes Implemenation

In this chapter we describe the design flow we followed in order to design and implement an FPGA-based reconfigurable node, executing the GPSR protocol. This approach consists of a state-of-the-art FPGA connected together with a low-power general purpose CPU board. This platform and the one that was described previously not only can comprise two different networks but can also be nodes of an heterogeneous wireless network, since their functionality is exactly the same.
An overall block diagram of the FPGA-based WSN node implementation is presented in figure 5.1. In this subsection we will describe the implementation details of this architecture.

The FPGA device utilized in our innovative design is the one from the Virtex 5

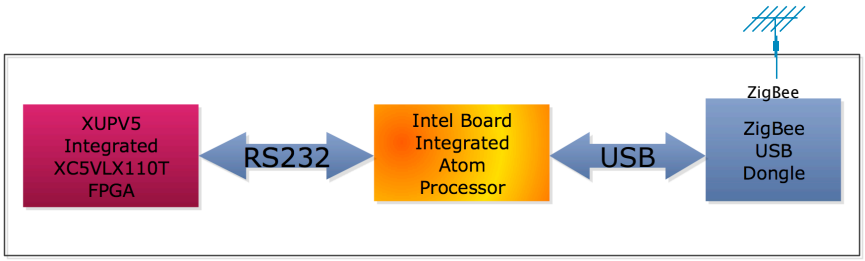

Figure 5.1: FPGA Node Block Diagram

family. The Virtex-5 family [95] FPGAs are the worlds first 65nm FPGA family fabricated in 1.0v, triple-oxide process technology, providing up to 330,000 logic cells, 1,200 I/O pins, 48 low power transceivers, and built-in PowerPC 440, PCIe endpoint and Ethernet MAC blocks, depending upon the device selected. They offer the best solution for addressing the needs of high-performance logic designers, high-performance DSP designers, and high-performance embedded systems designers with unprecedented logic, DSP, hard/soft microprocessor, and connectivity capabilities.

XUPV505-LX110T is a feature-rich general purpose evaluation and development platform with on-board memory and industry standard connectivity interfaces. It features a Virtex-5XC5VLX110T FPGA device supporting USB host, peripheral controllers, programmable system clock generator and many other I/O devices including RS-232 port [96]. The block diagram of the XUPV5 development board is presented in figure 5.2

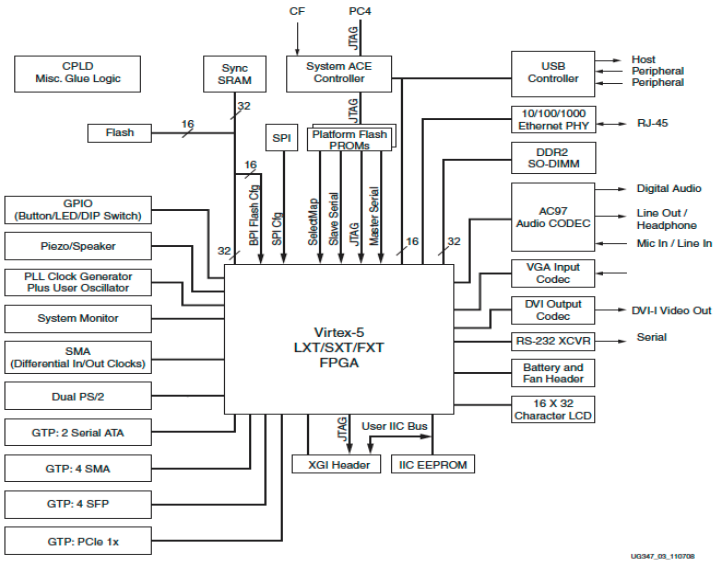

Figure 5.2: XUPV5 Evaluation Platform Block Diagram

5.1 GPSR Architecture

In this section we present the reconfigurable architecture of the GPSR Routing Protocol on an FPGA. The protocol was designed from scratch on a reconfigurable device, so as for the FPGA to act as a WSN node and to have exactly the same functionality as a common WSN mote. In the following subsection we present in detail this new proposed architecture.
The GPSR routing protocol was described in detail in chapter 3. Following this design specification, an overall block diagram of this protocol on an FPGA is presented

in figure 5.3. Our architecture consists of the following components:

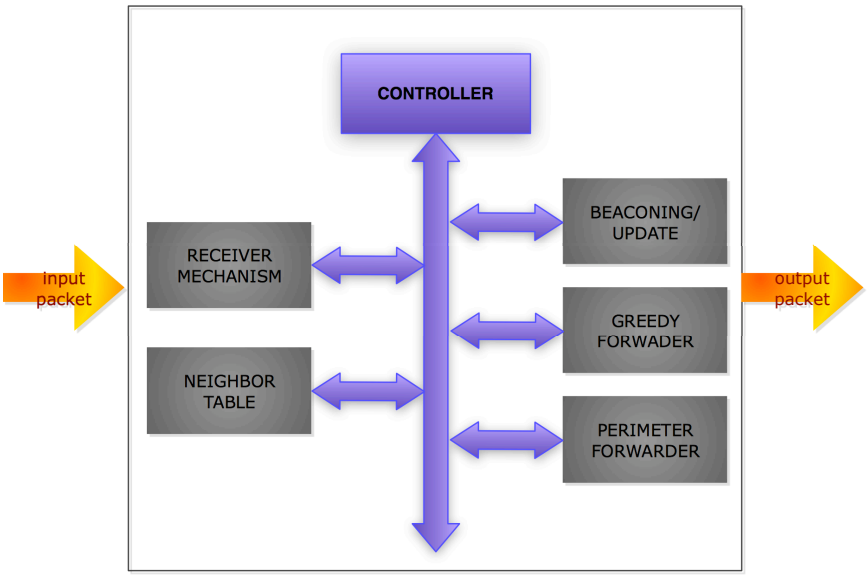

Figure 5.3: GPSR on FPGA Block Diagram

- **Receiver Mechanism**: It is responsible for receiving all incoming packets and, according to their type, send them to the appropriate module for further processing.
- **Neighbor Table**: This module stores all the neighbors of this node, with all the necessary information that is needed in order for the protocol to decide where to forward an incoming data packet.
- **Beaconing/Update Mechanism**: This module is responsible to receive and send beacon packets to the rest of the nodes of the neighbor, and update the neighbor table according to the incoming beacon packets.
- **Greedy Forwarder**: It implements the Greedy Algorithm, according to the GPSR specification.
- **Perimeter Forwarder**: It implements the Planarized Algorithm, so as to forward packets in cases where the Greedy Forwarder cannot decide where to forward an incoming packet.

- **Controller**: It arbitrates the functionality and the inter-communication of all the others modules

All the previously mentioned modules will be described later in the following subsections.
The reconfigurable GPSR (r-GPSR) takes as inputs a beacon or a data packet and subsequently its outputs are the same. The types of the packets are the same as the original GPSR and are presented in figures 3.9 and 3.10. The size of a beacon packet is 15 bytes, whereas the size of a data packet is 34 bytes. The input to r-GPSR has to be 35 bytes long (34 bytes the maximum length of a packet plus one indicating the packet's type). When a packet is of "beacon" type, it size is modified to be 34 bytes long. The unused fields remain zero, and the r-GPSR handles it as a beacon packet according to its type, that is added as an extra byte in a new field in the end of the packet. As far as the size of the data packets concerned (both perimeter and greedy), this is 34 bytes long plus one byte that is added in the software, which is their type. To summarize, the input of the r-GPSR is 35 bytes long or 280 bits, whereas the size of the output is also 35 bytes. The software is responsible to "add" or "cut" the unnecessary bytes, according to the type of each packet

5.1.1 Neighbor Table

The Neighbor Table architecture is presented in figure 5.4 and it consists of a controller, a Block RAM (BRAM) and some output registers. Single-port BRAM was chosen to be utilized for the neighbor table, which consists of 16 positions by 99-bits each. Every memory entry is presented in figure 5.5. The high 32-bits are used to store the ID number of each node, while the following 32 bits store the node's position (16 bit for the x co-ordinate and 16 bit for the y co-ordinate), assuming that all the nodes are placed in the first quartum of a 2-D Cartesian Coordinate System. The rest of the bits are used to store the status of each node.

The memory controller is an FSM which arbitrates the Neighbor Table and its functionality is the following: If the GPSR Controller asks all the entries of the the neighbor table, the memory controller signals the BRAM to read all them. In every cycle one position is read and all the memory's containers are stored to the registers, in order to be used later by every module that needs them. If a new entry

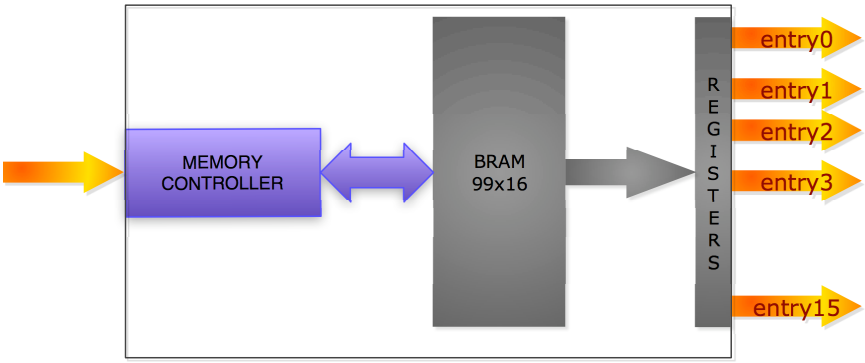

Figure 5.4: Neighbor Table Block Diagram

comes, meaning that a new node has been added to the neighbor, then the memory controller seeks for the first empty position in the neighbor table so as to store the new data. After that, the read process is repeated, so as all the modules to be notified for the new node. If an update notification is received by the GPSR controller, then the memory controller updates (update or delete) the neighbor table. The read process is repeated again, in the same manner as it was described previously.
Last but not least, we should mention that one of our assumptions is that in the first position of the neighbor table, the current node's ID, position and status is stored

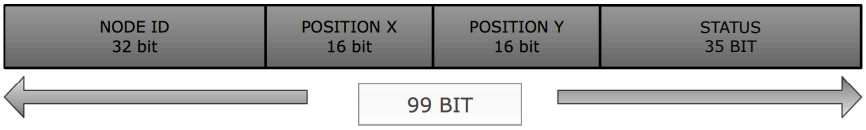

Figure 5.5: Memory Entry

5.1.2 Greedy Forwarder

The Greedy Forwarder module, as the name of the protocol implies, is the one that implements the previously described, greedy algorithm. It calculates the shortest path for one input packet and forwards it to the shortest node in the neighborhood table.

The Greedy Forwarder module is presented in figure 5.6 and consists of a module that finds the shortest path among all the nodes of the neighbor table using the Euclidean Distance Metric and some other components that are used to "fix" the new packet fields, according to the decision made by the Greedy Alogirthm.

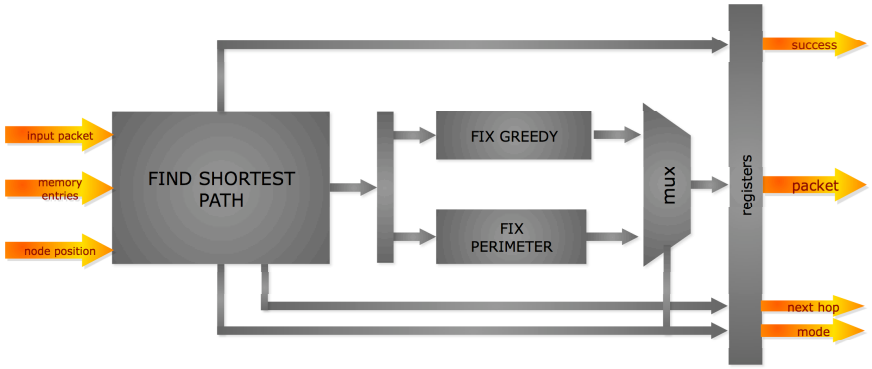

Figure 5.6: Greedy Forward Mechanism Block Diagram

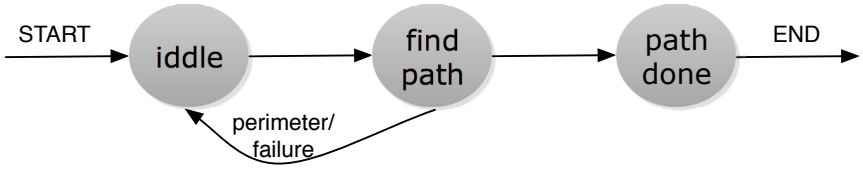

Figure 5.7: Greedy Forwarding Controller

This module takes as inputs the incoming packet (280-bits long), all the entries of the neighbor table and the current node ID and position. The outputs of this module are: a signal that indicates whether or not the calculation of the new path was successful (in other words if there is a shortest path for the packet to be transmitted through), the new packet with fixed fields, according to the path that it will be followed, the next node that the packet should be transmitted to and a signal that indicates whether or not this specific packet should remain in greedy or transit to perimeter mode. The last occurs when the "Find Shortest Path" mechanism fails to find a proper path.

The functionality of this module is arbitrated by a controller in the following way: Once the Greedy Forwarder module is enabled, it starts to find the shortest path among all the available paths. This is done by the "Find Shortest Path" module. This process is repeated for all the memory entries. This module returns, as it was previously described, the next node ID that a packet should be transmitted to. If it returns 0, it means that there is no greedy path for this specific packet, and thus perimeter forwarding should be tried. On the other hand, if there is a success in finding a shortest path, the packet fields are fixed and the packet is ready to be transmitted to the upper modules for further processing and transmission. The packet fields that are fixed are the destination node ID, which takes the value of the next node that the packet should be forwarded to and the new destination position of the target node. The rest of the packet fields remain the same and are copied from the incoming packet to the new one.

The "Find Shortest Path" module block diagram is presented in figure 5.8. In the

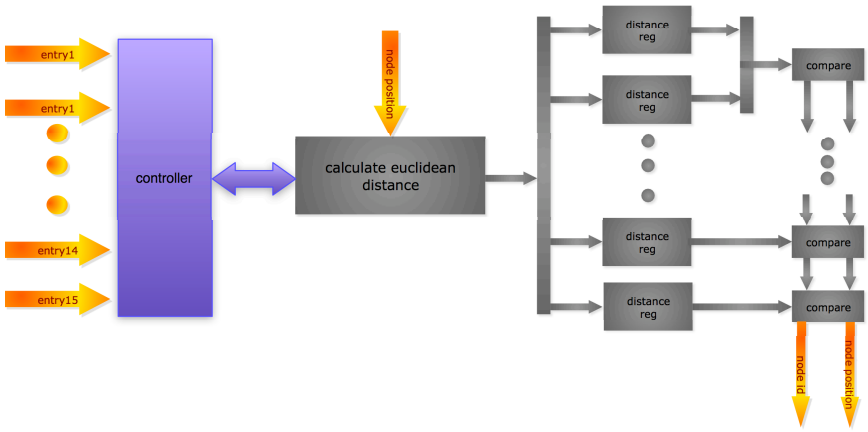

Figure 5.8: Find Shortest Path Block Diagram

"heart" of this module a circuit exists that calculates the Euclidean Distance among all the nodes' positions. It also consists of a controller which determines the way this module works, some registers to store the results and some comparators, which decide the final output of this circuit.

This module takes as inputs all the Neighbor Table entries as well as the current node position and ID. When this module finishes its process, will output the node position and the ID that the packet should be transmitted to.

Once this module is activated, it calculates in pairs, the distances between the node's position and every other node in the Neighbor Table. This process is done in a serial way, since it was necessary to minimize the resources that this specific circuit uses, despite the fact that in this way it becomes slower (this is not nevertheless the critical path of our architecture). All the calculated values are then stored into 32-bit registers, for further processing.
Uppon completion of the distances calculations, a successive pair-comparison for all the distances takes place. The first comparator has as inputs the two first distances. It compares them and outputs the lower, together with the lower distance's node ID and position. The second comparator takes as inputs the second calculated distance and the result of the previous comparator and finds the lower distance between them. This process is repeated for all the calculated distances. At last, the final comparator gives as result the total shortest distance, together with the node ID and the position of this specific node, which will be the new destination one.

The module that calculates the Euclidean Distance for all the paths is presented

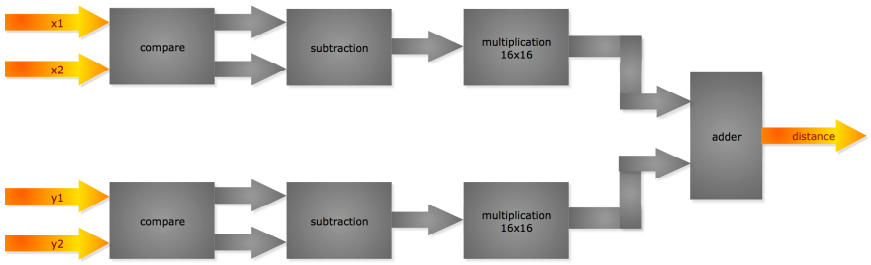

Figure 5.9: Euclidean Distance Block Diagram

in figure 5.9. It consists of two 16-bit comparators, two 16-bit subtracters, 2 16x16 multipliers and a 32-bit adder.
The Euclidean Distance Theorem was presented in the previous section of this chapter, so it will not be explained at this point.
This circuit takes as inputs 4 16-bit positive integer numbers, which represent the co-ordinates of two different nodes on a Cartesian Co-ordinate System. The x_1 and x_2 inputs represent the x-coordinates, while the y_1 and y_2 represent the y-coordinates of two different nodes. The output of this module is a positive 32-bit long integer number, which represents the Euclidean Distance, squared. At this point it should be mentioned that there was no need for the square root function to

be implemented, since the calculated numbers are used only for comparisons. So, the comparison result remains the same whether we compare two numbers or their square values.
The 16x16 multipliers are imported from the Xilinx Core Generator [87], while for the implementation of the subtracters, the comparators and the adder the VHDL Library was u utilized.
This module was designed to be fully combinational, which means that it outputs the result in one cycle. Firstly, the input values are compared so as to determine which is the greatest value. These numbers, in the proper order, are then used as inputs for the subtracters, so as to calculate their differences, which are then multiplied with themselves, so as to calculate their squares. Finally, these numbers are added together, which results to the distance between those co-ordinates.
In other words, the previously described circuit calculates the following mathematical statement:

$$Distance^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 \tag{5.1}$$

5.1.3 Perimeter Forwarder

The Perimeter Forwarder module tries to select a clockwise neighbor by using the "right-handed" rule and performing the "face-change" algorithm when a "greedy" neighbor is not found. It receives a perimeter-mode data packet, performs the "face change" algorithm and forwards a packet to counter-clockwise neighbor nodes. If the distance from local position to destination is shorter than one from previous node to destination it sends the packet to the greedy forwarder.
Figure 5.10 illustrates the Perimeter Forwarder block diagram. It consists of 3 major components: a module that tries to find a shortest path for the input packet (it was described in detail in the previous section), a module that implements the "Clock Wise Mechanism" (in other words it tries to find a clockwise path for a specif packet) and a module that implements the "face change" algorithm.
The Perimeter Forwarder module takes as inputs a data packet, which is of "perimeter" type, all the entries from the Neighbor Table and the local node position. The outputs of this module are: a signal that indicates whether there is a success in finding a perimeter path, the new packet when there was a success or the same packet when there was a failure, the next perimeter that the packet should follow and the mode that the packet should be in (greedy or perimeter). If the distance from local

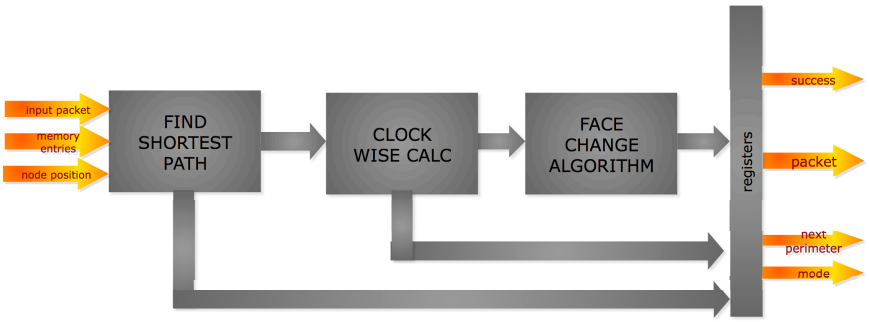

Figure 5.10: Perimeter Forwarder Block Diagram

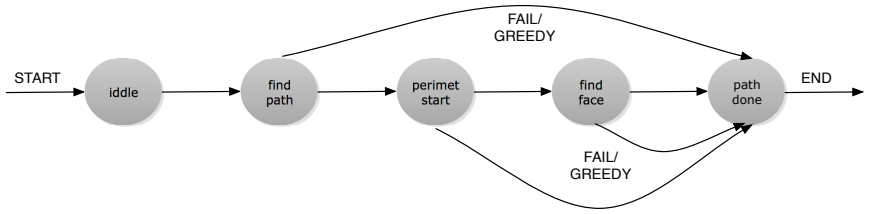

Figure 5.11: Perimeter Forwarder Controller

position to destination is shorten than one from previous node to destination, the new packet transits to greedy mode, else it remains in perimeter mode.

Figure 5.11 presents the FSM that was designed in order to control the Perimeter Forwarder module. When a packet in perimeter mode is received, it tries to find a shortest path for the given neighbor nodes. If it finds a greedy path, the Perimeter Forwarder finishes its execution, and the packet enters in greedy mode. In the opposite case, the module tries to find the best perimeter in clockwise order so as to forward the packet. Once a path is found, the "Face Change" algorithm starts its execution in order to determine whether the packet has reached an edge that crosses the line to the destination. The Face Change returns the appropriate next hop for the first edge on the next face, if such a crossing edge has been reached, or the unchanged next hop found by the right-hand rule if no crossing of the line to the destination is found.

The module that is used to find a clockwise path for a perimeter packet is the "Find Clock Wise Path" one and is presented in figure 5.12. The packet is forwarded to

the first edge in clock wise order from the current node using the right hand rule. So, this module finds all the paths that a perimeter packet could be traversed from.
It takes as inputs a Neighbor Table entry (node ID, location and status) and the

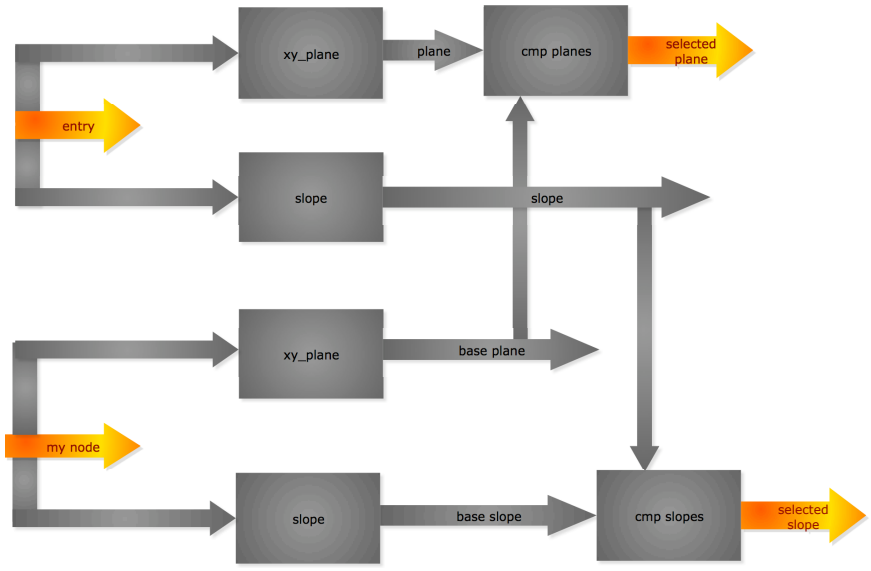

Figure 5.12: Find Clock Wise Path Block Diagram

local node data (position and ID).Its outputs are the selected slope and the plane that this packet may follow. It consists of 2 "xy plane" modules that try to find the plane that this packet should follow and 2 modules that calculate the "base slope" and the slope that this packet may follows respectively. It also consists of 2 modules that compare a base slope with a calculated one and a base plane with a calculated one, and decide, given some rules, which are the best to follow. The rules, that the comparisons are made are described in detail in the Software Documentation of the GPSR Routing Protocol [70].
The "xy-plane" module architecture is presented in figure 5.13. It takes as inputs 4 16-bit values which represent the coordinates of two nodes and it outputs the best plane based on the following comparisons: If $x1 > x2$ and $y1 > y2$, the output is 1 in 32-bit representation. If $x1 <= x2$ and $y1 > y2$, the "xy plane" is 2, if $x1 < x2$ and $y1 <= y2$, "xy plane" is 3 and finally if $x1 >= x2$ and $y1 < y2$, "xy plane" is 4.
The "Slope" module block diagram is presented in figure 5.14. It is used to calcu-

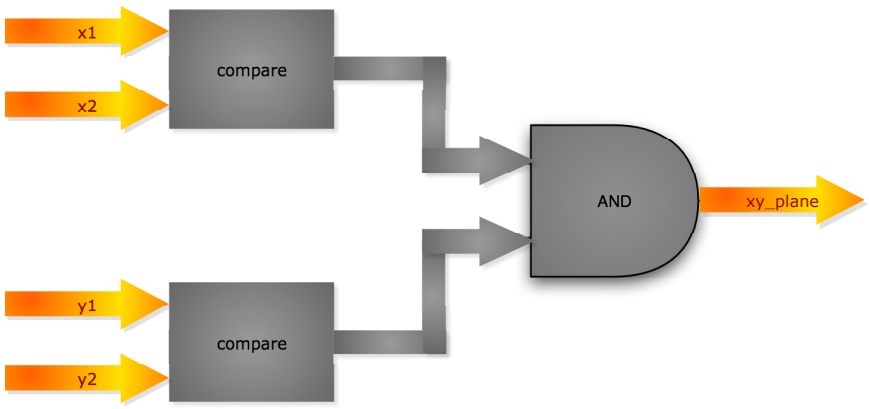

Figure 5.13: XY Plane Block Diagram

late the slope of 2 given coordinates (x_1, y_1) and (x_2, y_2). These values are 16-bit unsigned integers. The outputs of this module are two numbers indicating the value of a slope. This value is represented by 2 16-bit unsigned values (the remainder and the quotient of a division).
The Slope module works as follows: The input co-ordinates are subtracted ($x = x_1 - x_1$, $y = y_1 - y_2$) and the results are divided according to the following formulas:

$$quotient = X/Y \tag{5.2}$$

$$remainder = X\%Y \tag{5.3}$$

For the implementation of the 16-bit subtracters, we used the VHDL library templates, whereas for the implementation of the divider we utilized the Core Generator. Regarding the divider [88], it is based on Radix-2 non-restoring division. The Radix-2 algorithmic exploits fabric to achieve a range of throughput options, including single cycle. Since the result of the division is not ready in one cycle, the controller of the slope delays the output of this module waiting for the completion of the division.

Regarding the "Face Change" module, it is presented in figure 5.15, whereas the pseudo code for this algorithm is presented in 5.1.3. This module is responsible for performing the "Face Change" algorithm, which determines whether the packet has

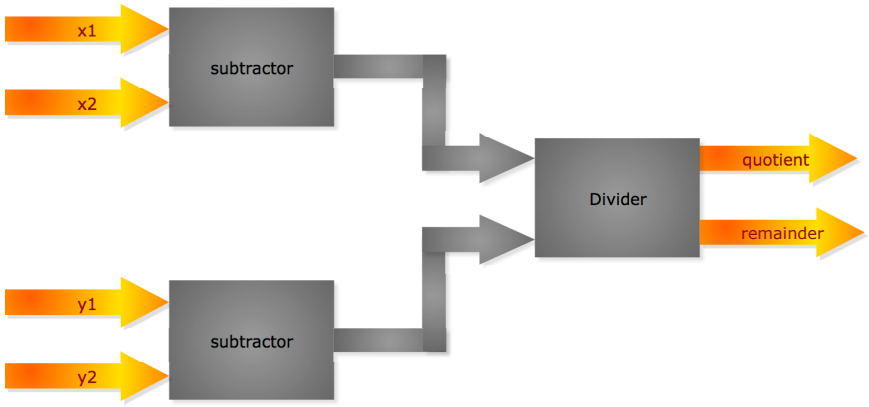

Figure 5.14: Slope Calculation Block Diagram

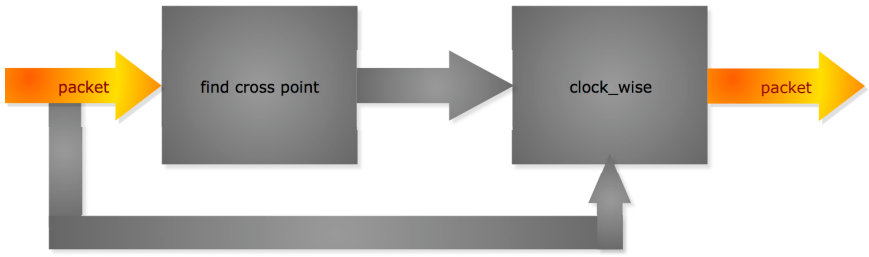

Figure 5.15: Face Change Algorithm Block Diagram

reached an edge that crosses the line to the destination. It returns the appropriate next hop for the first edge on the next face if such a crossing edge has been reached, or the unchanged next hop found by the right-hand rule if no crossing of the line to the destination is found. Note that the changing faces amounts to treating the next hop on the current face as the *previous hop*, and applying the right-hand rule. Face-change calls itself recursively, because it is possible that a single node borders multiple edges that cross the line to the destination. The recursion terminates upon reaching the edge that crosses the line at the closest point to the destination; it must terminate because there is always an edge that crosses the line at a point farther than this closest point.

As figure 5.15 clearly depicts, this module consists of two major components. A module that finds the previously mentioned cross points and a module that finds

Algorithm 1 Face Change Pseudo Code Algorithm

$FACE - CHANGE(p, t)$
$i = INERSECT(t.l, self.l, p.Lp, D)$
if $i \neq NIL$ **then**
  if $DISTANCE(i, D) < DISTANCE(p.Lf, D)$ **then**
    $t = RIGHT - HAND - FORWARD(p, t)$
    $t = FACE - CHANGE(p, t)$
    $p.e_0 = (self.a, t)$
  end if
end if

the clock wise path for this specific cross point and packet. It takes as input the incoming perimeter packet and outputs the new perimeter packet with updated fields, according to the path that this packet should follow.

The "Find Cross Point" module is presented in figure 5.16 and its main function-

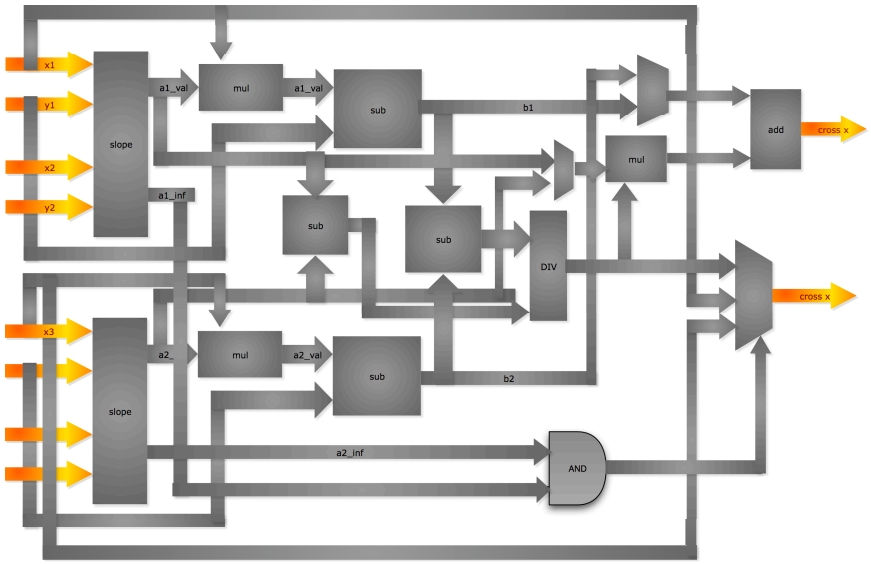

Figure 5.16: Find Cross Point Block Diagram

ality is to find the crossing points of the planarized graphs when the face change algorithm is performed. It takes as inputs the location in which the packet entered in perimeter mode (L_p), the location the packet entered the current face (L_p), the po-

sition of the local node and a calculated position that was calculated initially by the clock wise mechanism, when the perimeter forwarding mechanism started its execution. This module will output the location where a crossing point $(cross_x, cross_y)$ exists. It consists of several components, the major of which are 2 "slope modules", for the calculation of the slopes of the given locations, 3 16x16 multipliers, 4 subtracters, divider and an adder. The output of this packet are 2 32-bit unsigned integer values.
This module works as follows: First the slopes are calculated for the given locations. When slopes are calculated the remainder values are checked so as to determine if they are equal or not. If they are both equal to zero, the quotient values are compared together. If they are equal the process ends and is repeated for other input combination. If they are not equal the b_1 and b_2 values are calculated from the following formulas:

$$b_1 = pos_{y1} - a1_{quotient} * pos_{x1} \tag{5.4}$$

$$b_2 = pos_{y3} - a2_{quotient} * pos_{x3} \tag{5.5}$$

Uppon completion, the division takes place, with dividend the subtraction result b_1 from b_2 and divisor the subtraction result $a1_{quotient}$ from $a2_{quotient}$. The result of the division is the $cross_x$. Afterwards, the $croos_y$ value is calculated from the following formula:

$$y = a1_{quotient} * x + b_1 \tag{5.6}$$

If $a2_{remainder}$ is equal to zero and $a1_{remainder}$ is not, the $cross_x$ takes the value of the x_1 co-ordinate, while the $cross_y$ takes the value of the following formula:

$$y = a2_{quotient} * cross_x + b_2 \tag{5.7}$$

Finally, if $a1_{remainder}$ is equal to zero and $a2_{remainder}$ is not, then the $cross_x$ takes the value of the x_3 co-ordinate, while the $cross_y$ takes the value of the following formula:

$$y = a1_{quotient} * cross_x + b_1 \tag{5.8}$$

In all the previously described cases, a check is executed after the calculation of all the crossing points to decide whether there was a success in finding a cross point or not.

5.1.4 Update Mechanism

The update mechanism is used so to find which nodes inside the Neighbor Table has to be updated (position, status). Updating means either to update an existing entry, to add a new entry or delete one. Figure 5.17 presents an abstract block diagram of this module.

It takes as inputs all the entries from the Neighbor Table and the received packet

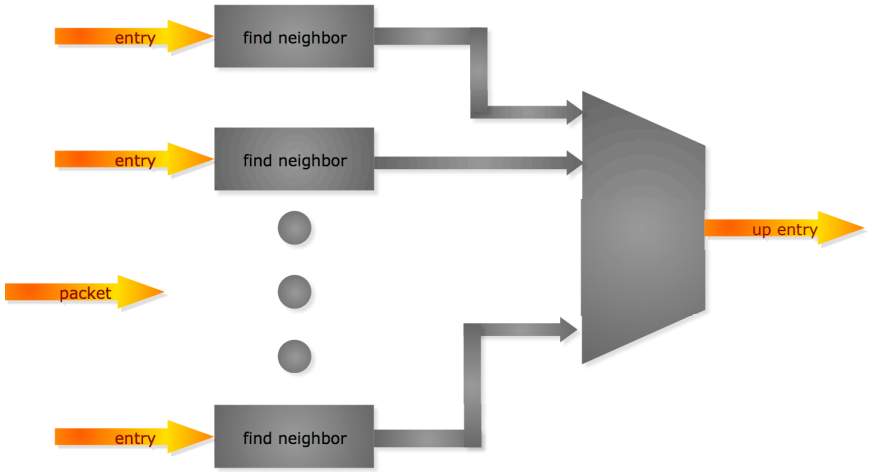

Figure 5.17: Update Mechanism Block Diagram

and outputs the updated entry with some extra control signals that indicate either the adding the updating or the removal of a node from the Neighbor Table.
16 simple comparators are used to compare every entry's ID with the input one so as to decide if the node exists or not. After that, this entry is updated and sent back to the Neighbor Table.

5.1.5 Trust Mechanism Architecture

The Trust mechanism is used to calculate the trustness between the current node and all its neighbors. It consists of a module that calculates the Direct Trust Value, one the calculates the Indirect Trust Value and a Total Trust Calculator. In this section we describe in detail the architecture of this mechanism always according to the description that was presented in the previous chapter. The main goal is to

make the r-GPSR protocol to operate in terms of security and trustness.
At this point it should be mentioned, that since this trust model has never been implemented in the past, either in software or hardware, the provided details were not sufficient in order to implement this module. To be more specific, we don't have the arithmetic details of the values used as well as the values of the constants. As a result, we designed a more abstract architecture, which, on the other hand, keeps up with the proposed trust model.

The first module that the Trust Mechanism consists of, is the Direct Trust one,

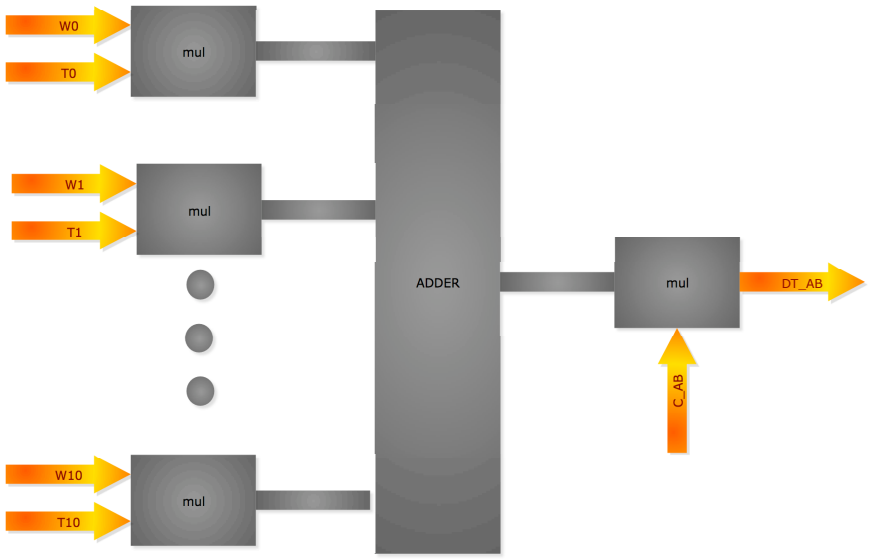

Figure 5.18: Direct Trust Between A and B nodes Block Diagram

which is presented in figure 5.18. A node's A Direct Trust value for its neighboring node B, i.e. $DT^{A,B}$ with k even types, can be calculated according to the following equation:

$$DT^{A,B} = C^{A,B} * (\sum_{i=1}^{k} W_i * T_i^{A,B}) \tag{5.9}$$

where W_i is the weighting factor for each one of the k event types and $T_i^{A,B}$ is node's A trust value of event i regarding node B.
It consists of 10 multipliers, if we assume that the number of nodes in the Neighbor

Table is 10, an adder, and a multiplier which calculates the Direct Trust value between A and B nodes after multiplying the Confidence factor with the previously described value.

Regarding Direct Trust Table that was mentioned in the previous chapter, this is

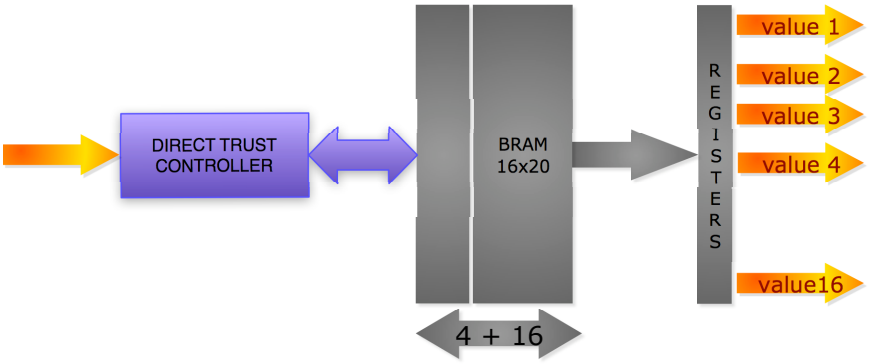

Figure 5.19: Direct Trust Table Block Diagram

presented in figure 5.19. This table is used to store the most important information so as for the Direct Trust value calculation to be feasible. It consists of a single port BRAM that is organized by 16 entries of 20 bits each. The first 4 bits are used to store a number that corresponds to the Direct Trust attributes and the next 16 are used to store the value of each attribute.

For each one of the of the Direct Trust Table node's A Trust value regarding node B, i.e. $T_i^{A,B}$, can be calculated as follows:

$$T_i^{A,B} = \frac{a_i S_i^{A,b} - b_i F_i^{A,B}}{a_i S_i^{A,B} + b_i F_i^{A,B}} \tag{5.10}$$

where:

- $S_i^{A,B}$ is the number of successful type i events that A has measured for B
- $F_i^{A,B}$ is the number of failed type i events that A has measured for B
- a_i and b_i represent the weight/significance of a success vs. the weight/significance of a failure of type E_i events.

The block diagram of the previous function is presented in figure 5.20. It takes as inputs the a_i, S_i, b_i and T_i values and outputs the T_{AB} value. For this module two multipliers, a subtracter, an adder and a divider were utilized.

As fas as the Confidence Factor C_{AB} concerned, this is calculated based on the

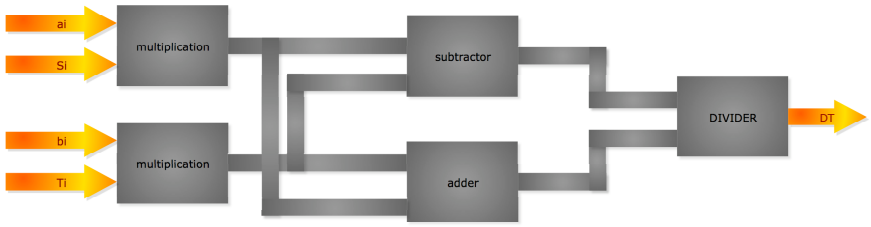

Figure 5.20: Trust A B Block Diagram

following formula:

$$C^{A,B} = 1 - \frac{1}{noi + a_{10}} \tag{5.11}$$

where noi indicates the number of interactions with node B and a_{10} is a factor whose value will be checked during simulation testing. This confidence factor can be proved useful, especially during the beginning of network operation. Moreover, in case of GPSR, a proper metric that can be implemented in this trust model which is the distance of each one of the neighboring nodes to the sink. The closer a node to the sink, the greater the value added to the final direct trust of the node.
The block diagram of the Confidence Factor is presented in figure 5.21 and utilizes an adder, a subtracter and a divider. The Indirect Trust block diagram is presented in figure 5.23. The designed architecture is based on the following equation:

$$IT^{A,B} = \sum_{j=1}^{n} W(DT^{A,N_j}) * DT^{N_j,B} \tag{5.12}$$

where n is the number of neighboring nodes to A, N_j are neighboring nodes to A, $DT^{N_j,B}$ is node's N_j reputation value of node B and $W(DT^{A,N_j})$ is a weighting factor reflecting node's A direct trust value of node N_j. Different weighting factors are used for each node regarding the events described above. For example, if node's C direct trust value (evaluated by node A) is large and also node C is frequently sending responses to node's A requests, then its weighting factor is large. It consists of several number of dividers and multipliers (their number is equal to the number

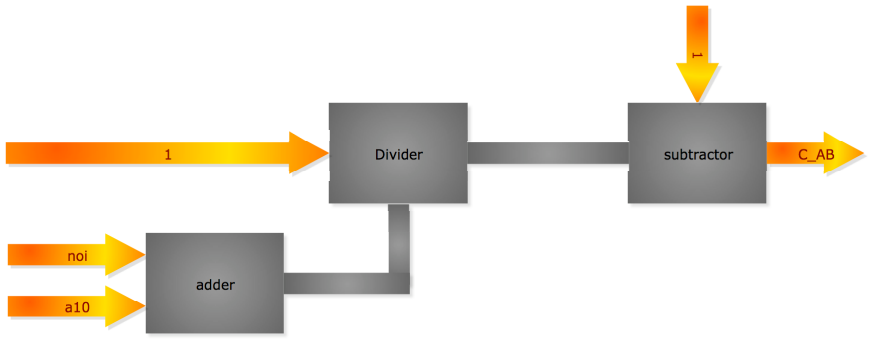

Figure 5.21: Confidence Factor Between A and B

of nodes in the Neighbor Table), and an adder which adds the previously calculated values together so as to output the Indirect Trust Value.

The Indirect Trust Table is organized as presented in figure 5.23. This table

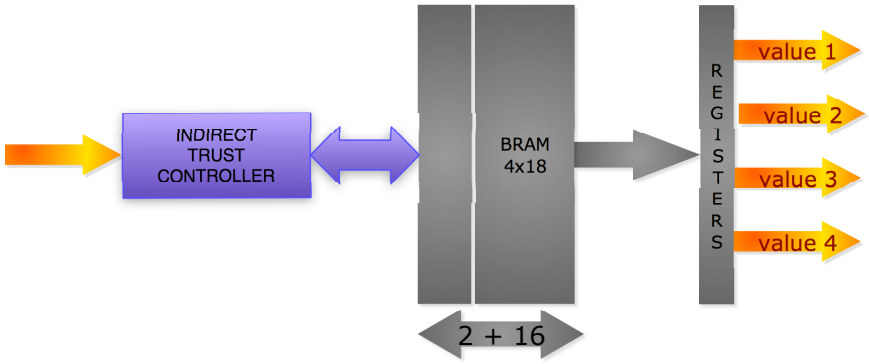

Figure 5.22: Indirect Trust Table Block Diagram

contains all the attributes that were described in the previous chapter.

The total trust evaluation node A of node B, i.e. $TT^{A,B}$ is performed by applying the following equation:

$$TT^{A,B} = W(DT^{A,B}) * DT^{A,B} + W(IT^{A,B}) * IT^{A,b} \tag{5.13}$$

where $DT^{A,B}$ is node's A trust value of node B, $W(DT^{A,B})$ is a weighting factor reflecting node's A direct trust value of node B, $IT^{A,B}$ is a node's A indirect trust

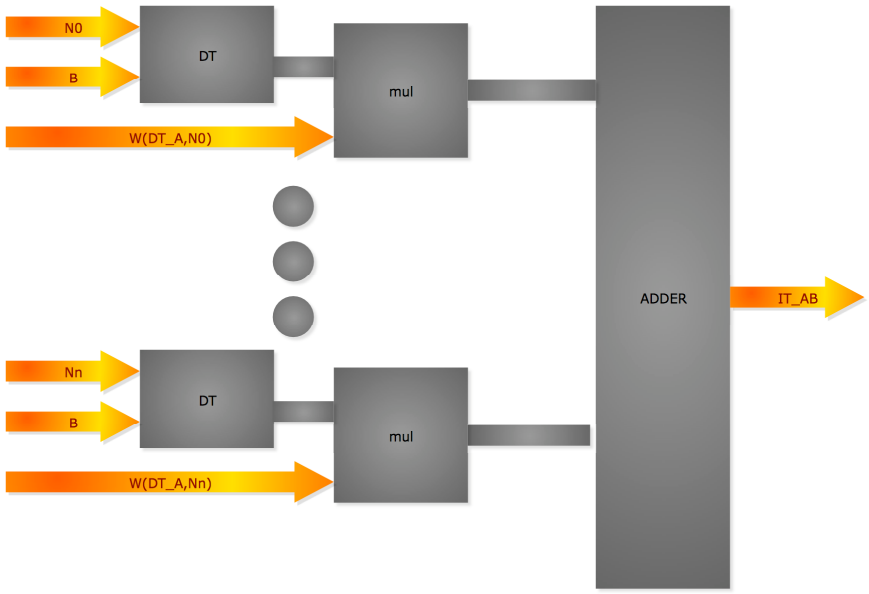

Figure 5.23: Indirect Trust Between A and B nodes Block Diagram

value of node B and $W(IT^{A,B})$ is a weighting factor reflecting node's A indirect trust value of node B. The block diagram of the Total Trust Module is presented in figure 5.24. It contains the previously described Direct and Indirect Trust modules, 2 multipliers and an adder that calculates the Total Trust Value. Every time that the cooperation among the nodes is completed, every node records and updates the trust value of its cooperation node. The trust value of a node varies with time. In our trust model, if a node cannot provide cooperation for other nodes, the other nodes will gradually decrease its trust value accordingly. Since node A can be sure only about the first-hand information that has collected, the weighting factor of the Direct Trust Value will be larger than the weighting factor of the Indirect Trust value. This remark might not be applicable in the case where a new node appears in the neighborhood, where indirect information may be the only source of information to be used for validating the neighbor's trustworthiness. Regarding the embedding of the Trust Mechanism with the rest of the r-GPSR architecture, this is presented in an abstract way in figure 5.25, but we have not implemented this actually. We only present here the way that this module can be added on our r-GPSR architecture that will transform it to Trusted and Secured GPSR protocol.

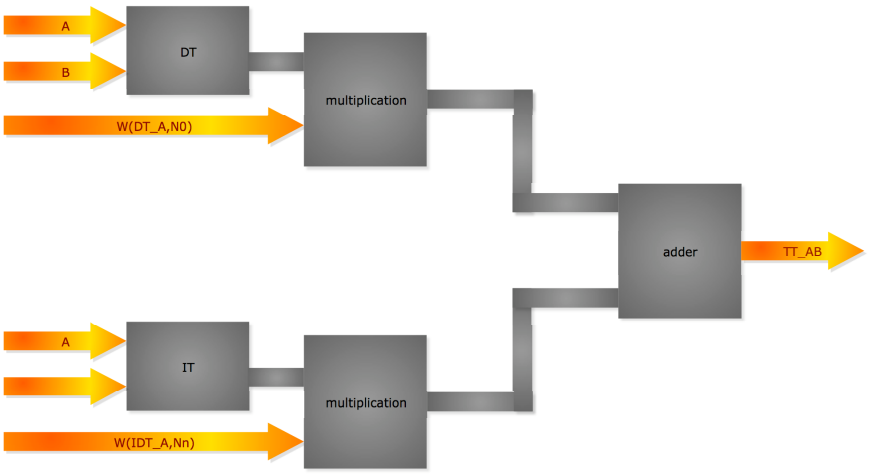

Figure 5.24: Total Trust Between A and B nodes Block Diagram

5.1.6 GPSR Controller

In order to make all the components that comprise the r-GPSR protocol to communicate and co-operate, an effective controller was necessary to be designed. Figure 5.26 depicts the FSM controller that was designed for this purpose.
This FSM consists of 10 states, at one of each a specific operation of the GPSR is triggered. Initially, the FSM is at the idle state. When a packet arrives, the Neighbor Table is activated and all the existing entries in it are read and stored into registers. After that, the type of the packet is checked, in order to be determined whether this specific packet is beacon or data.
If it is a beacon packet, the FSM transits to the "Beacon Receive" state. The appropriate fields are read, the "Update Mechanism" is activated and upon completion, a new Beacon Packet is ready to be send to the output, with location co-ordinates and ID, the current location and ID of the local node respectively.
If the packet that was received is a data one, the FSM transits to the "data receive" state, where is determined whether it is "GREEDY" or "Perimeter" one. At this state, the Update Mechanism is also triggered in order to update, if necessary, the Neighbor Table.
In case of a "GREEDY" packet, the FSM waits for the Greedy Forwarder mechanism to complete and send the packet to the output. If that is not possible, the control passes to the Perimeter mechanism, so as to find out whether a possible

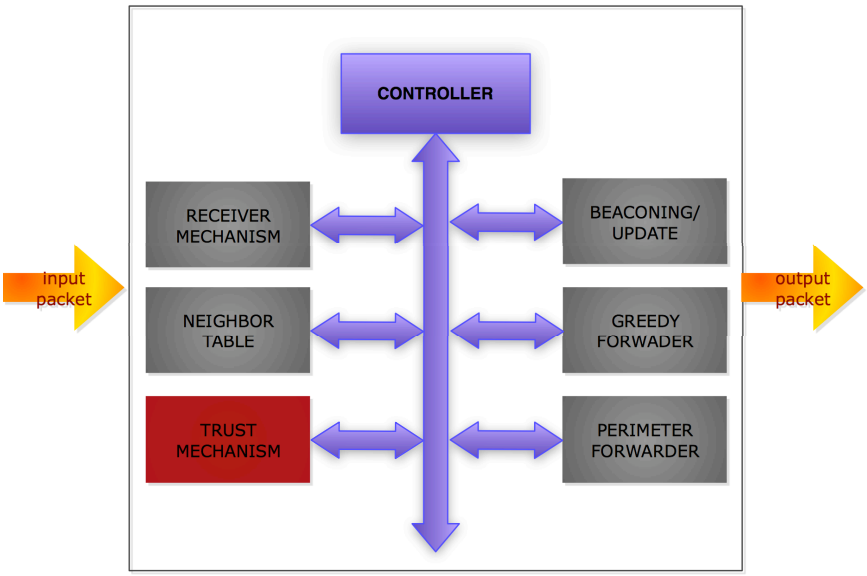

Figure 5.25: Trusted GPSR on FPGA Block Diagram

perimeter exists for this specific packet and neighbor nodes stored into the neighbor table. If yes, the new packet is a perimeter one, else a copy of the incoming packet is sent back to the upper levels (which will be described in the next chapter) with the appropriate failure code.
Finally, in case of a perimeter packet, the same process is repeated respectively.

5.2 Implementation Details

The whole GPSR protocol architecture was implemented on this device. As far as some implementation details concerned, for the Neighbor Table's memory, we used single port BRAM, while the initialization of this memory was made with the use of a ".coe" file, which contains some initial nodes' IDs together with their position and status data. At this point it should be mentioned that, the node's positions, when GPSR is used, have to be pre-defined when there is no GPS receiver. The tool used to implement our design was Xilinx ISE 10.1 [92], while its embedded simulator was used in order to verify the correct operation of our architecture via the process of "Behavioral Simulation". Next, we had to carry out "Post Place and

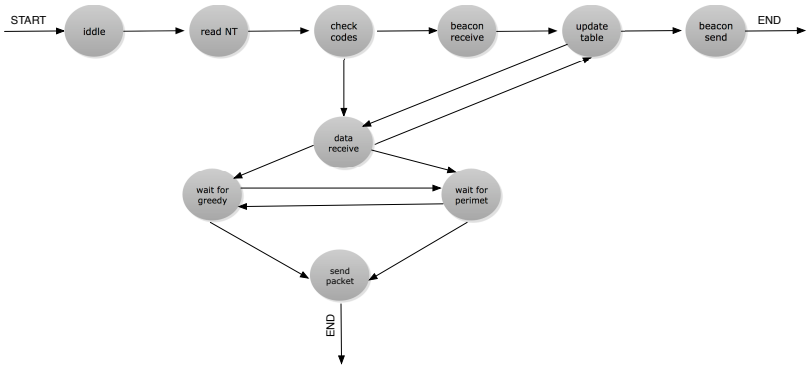

Figure 5.26: r-GPSR Controller

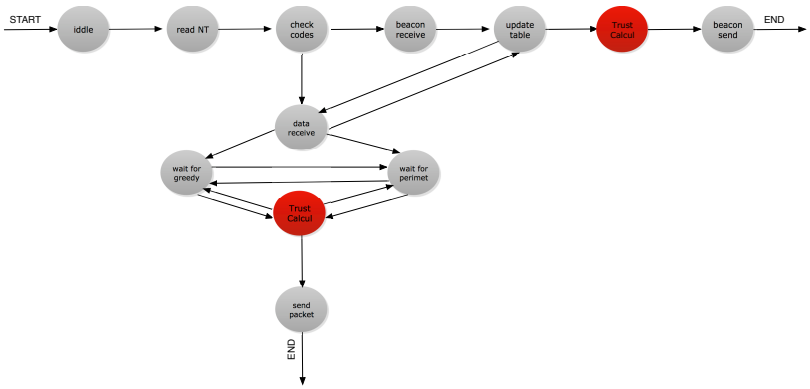

Figure 5.27: Trusted r-GPSR Controller

Route Simulation" and, for this purpose, we preferred Modesim SE 6.3f [93]. The FPGA was programmed using the embedded Xilinx ISE tool.
The previously described device is connected with an Intel Desktop Board D945GCLF2 that contains an integrated Intel Dual-Core Atom 330 processor @ 533 MHz [97]. The Intel Desktop Board D945GCLF2 is designed to support Internet-centric computing in a Mini-ITX form factor using the Intel 945GC Express Chipset. Besides that, our board is equipped with a 1-GB DDR2 RAM module and an external SATA2 hard disk drive. The block diagram if this device is presented in figure 5.28.

The connection of the previously described devices was made via RS-232 port. We connected the development systems with a null modem serial cable, utilizing

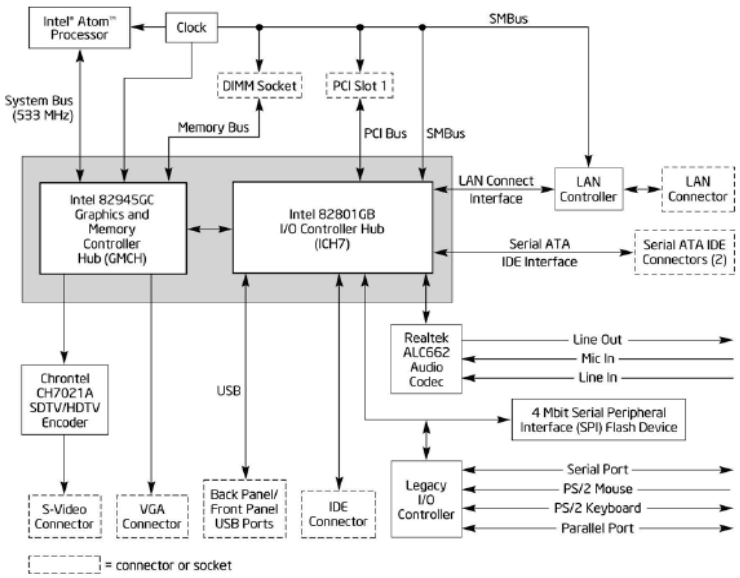

Figure 5.28: Atom Board (D945GCLF2D) Block Diagram

the RS-232 ports of each device @ 115200 Kbps., since the GPSR routing protocol doesn't require high data transfer speed. As far as the wireless part of our innovative platform concerned, we used the Crossbow MIB520CB USB Gateway connected to the Intel Atom Board.

As it was previously described, the communication between the Intel Atom board and the XUPV5 is facilitated through the RS-232 ports. So, we have designed and implemented a simple RS-232 interface in the reconfigurable device. The reconfigurable UART module consists of a simple serial receiver, a BRAM, which is used in order to temporary store the input and output data, a simple serial transmitter, and a simple controller, which arbitrates the functionality of the previous modules (Appendix A).

Figure 5.29 shows the top view of our platform. In this figure the FPGA used in inside the red frame, the ZigBee module inside the yellow one and the Atom processor inside the blue one. The rest parts are the prototyping areas used mainly for the intercommunication and the development of the platform.

Apart from the hardware modules utilized in our system, a software suite was also developed in order to enhance our platform with the appropriate functionality. To

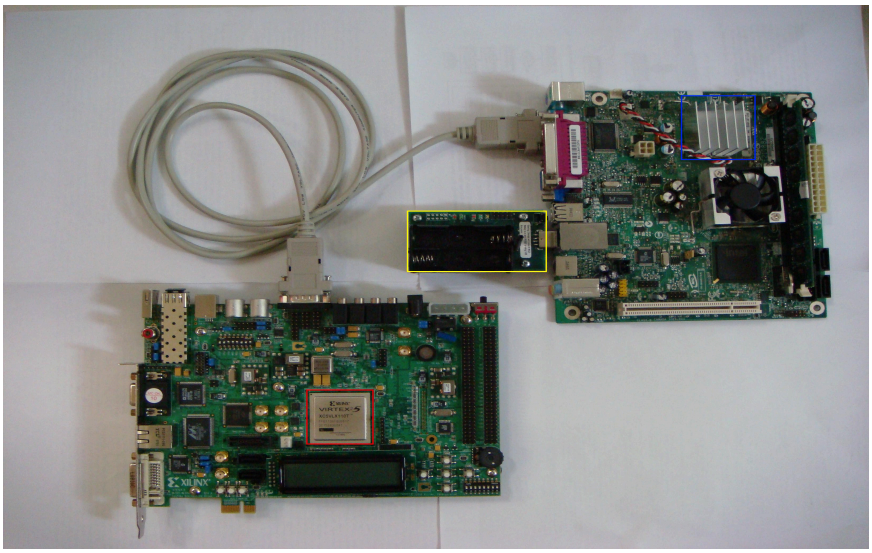

Figure 5.29: FPGA Node Top View

begin with, one of the most crucial issues was the correct selection of the operating system of the the Intel Atom Board. The operating system should be as minimal as in can, so as to meet the WSN need for lower power consumption; as a result, Linux Xubuntu 8.10 with Linux kernel 2.6.27 was selected, which is claimed to be appropriate for low power solutions [98].
The ZigBee module that is connected on a USB port of the Atom Board consists of an MIB520CB gateway and an IRIS mote connected together. It is programmed with a TinyOS utility application called *"BaseStation"*. This application acts as bridge between the serial port and radio network. When it receives a packet from the serial port, it transmits it on the radio; when it receives a packets over the radio, it transmits it to the serial port. Because TinyOS has a tool chain for generating and sending packets to a mote over a serial port, using a *"BaseStation"* allows PC tools to communicate directly with mote networks.
A python software suite was developed on the top of the Xubuntu OS which controls the efficient and correct data transfer among the FPGA and the ZigBee interface. Python [99] is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools and comes with extensive standard libraries. For

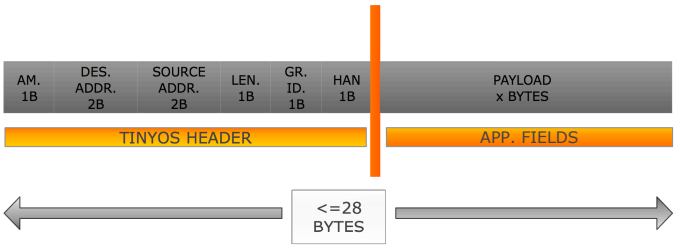

Figure 5.30: Serial Forwarder Message Format

our application, we used the open-source PySerial, Socket and MySQLdb, python libraries. PySerial is a library which provides support for serial connections over a variety of different devices: old style serial ports, Bluetooth dongles, infra-red ports, and so on. In our case, PySerial provides all the necessary functions for the com-

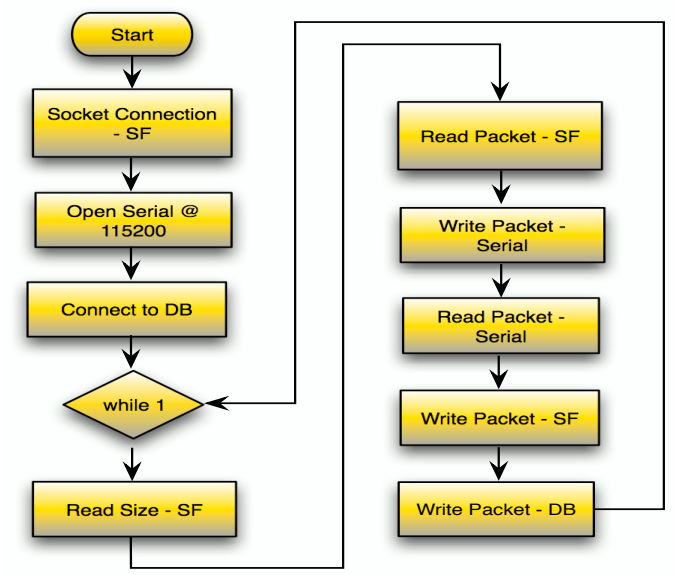

Figure 5.31: Python Script Flowchart of FPGA Node I/O Communication

munication between the Atom Processor and the FPGA Board, whereas the Socket library provides access to the BSD socket interface. The Socket Python library is also utilized for the interconnection with the ZigBee module, through the SerialForwarder Tool, which is described later in this section. Last but not least, MySQLdb

is a thread-compatible interface to the popular MySQL database server that provides the Python database API, which enables us to store our results directly to a database for easier retrieval and post-processing.

The SerialForwarder program opens a so called packet source and let many appli-

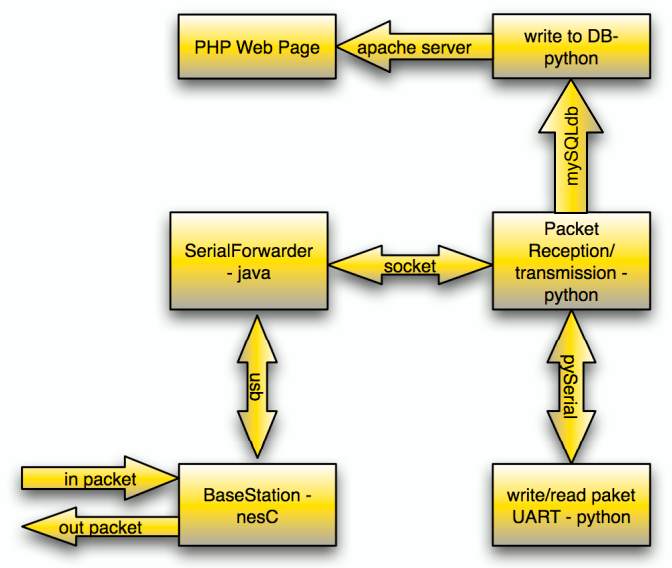

Figure 5.32: FPGA Node Software Stack

cations connect to it over a TCP/IP stream. For example, a SerialForwarder whose packet source is the serial port can be executed; instead of connecting to the serial port directly, applications connect to the SerialForwarder, which acts as a proxy to read and write packets [100]. Figure 5.30 presents the message format that the Serial Forwarder can recognize.

In general, our new node is used to receive, process, forward and optionally store data packets using all the previously referred software tools. Figure 5.31 presents the flowchart of our basic software suite while Figure 5.32 presents the complete software stack of our development tools. Utilizing our development tools, any application implemented in the Atom CPU, the FPGA or a combination of the two, can easily process incoming packets from the Motes. Upon a packet is received, our suite reads its size and the actual data itself. The received packet, then, is either processed by the software executed on the Atom or written to the RS-232 port, in

order to be processed by the FPGA. When an incoming packet is monitored in the RS-232 port, our software suite reads it, forms it according to the SerialForward Protocol and sends its size and the actual data to the socket, so as to either be further processed by the software executed on the Atom or be broadcasted to the air via the ZigBee module. Our development tools can also store this packet to a database.

Regarding the implementation of the previously described trust metrics, it should be noted that we have implemented them stand alone on a XUPV5 device. We haven't though connected the implemented module with the rest GPSR module, since we are still unaware of some important details such as the trust metrics arithmetic. Also the software implementation is still (when we write down these lines) under development and consequently we cannot verify the GPSR based on Trusted attributes.

Chapter 6

Reconfigurable (FPGA) BaseStation Implementation

The hardware modules used for the implementation of the Base Station node are the same as for the FPGA-Based node: A XUPV5 Development platform, an Intel Atom Development Board and a Zigbee Module. The FPGA device is programmed with the Base Station architecture that was described in the previous chapter. The software stack remains almost the same. The only difference is that our node does not send any data packets back to the air; only beacon packets are broadcasted through the Zigbee Dongle. Regarding the Data packets, these are aggregated in the Basestation node, and are stored to a database for further processing.

6.1 Architecture

In this section we describe the architecture of the r-GPSR routing protocol on a Base Station Node, which includes an FPGA Device. The previously proposed r-GPSR protocol was modified in order for the FPGA device to act as a BS.
In fact, the protocol functionality remains the same, as it is presented in figure 5.3, but with less functionalities.
A BS is usually the final destination for all the packets that are transmitted from all the nodes. It has only to receive beacon messages from the nodes inside its neighborhood, to notify its neighbors about its existence and to receive and store data packets. Consequently, the modules that are necessary for the FPGA to act as a BS are only a Neighbor Table, the Beacon/Update module, a Receiver Mechanism and a controller to arbitrate the module inter-communication.
The Neighbor Table architecture remains the same, as it was described previously: It is responsible to keep the states of all the node inside the neighborhood.
The Receiver Mechanism is slightly modified it order to receive data packets and to notify the Software Stack so as to store the data included in these packets.

The Beacon\Update module is responsible to generate and broadcast beacon messages to the rest of the neighbors and update correctly the Neighbor Table with the nodes' states.
The Greedy and Perimeter Forwarders are not used, since no data packets can be transmitted by the Base Station Node.

Chapter 7

System Verification, Monitoring Tools and TestBed

In this chapter we present the verification processes that were followed during the systems development. We also give some examples of monitoring tools that were used in order to view the ZigBee exchanged packets. Finally, we present a testbed that can be used for exhibiting our systems.

7.1 System TestBed and Verification

In this section we describe all the steps we followed in order to verify our proposed platforms. We describe in detail the verification process for every module separately but as a whole system too. Also, we present some scenarios that we have been examined and some other applications that can be used on this platform.

7.1.1 Sub-systems Verification

Our proposed systems have been tested and verified in every step of the development:

- **CPLD Architecture**: The architecture that was implemented on the CPLD device, was first simulated with the ModelSim Simulator via the "Behavioral Simulation" process. The results of the "Euclidean Distance" were compared and verified with those that came out of a simple Matlab Script which calculates the distances for several inputs. Also "Post-Fit Simulation" process was conducted, in order to examine the functionality of our module right before this is downloaded on the CPLD device.
- **CPLD-Mote Communication Protocol**: For the verification of the CPLD-Mote communication protocol we implemented the following scenario: First we implemented the communication protocol on the CPLD, without any other

functionality except for that the CPLD should send back to the Mote the inputs inverted. We implemented also in nesC an application that increases a counter, sends these values to the CPLD through the custom-made cable, takes the inverted inputs from the CPLD and broadcasts the initial counter value and the inversed one to the air. With the use of the ZigBee dongle we capture all these packets and verify them.

- **CPLD-Based Node**: In order to verify the correct functionality of the CPLD-based platform, we programmed the Mote with the modified GPSR Software that supports the communication protocol and has a specific location and an ID. The CPLD was programmed with the "Euclidean Distance" module and the CPLD communication protocol. This node acts as a GPSR network node. First, we verified that this node sends GPSR packets to the air. Afterwards, we checked, with a use of a ZigBee Dongle, whether this node forwards packets to other nodes or not. Finally, we programmed another mote with the same ID and location attributes, and for the same coordinates and number of nodes in the network, we compared if the node forwards the packets in the same manner as the new one. We also used for our experiments MICAz and IRIS motes in order to examine if the behavior of our new system remains the same.

- **FPGA Architecture**: All the modules that comprise the r-GPSR architecture were simulated separately with the "Behavioral Simulation" process with the use of the ModelSim Simulator. All the results were examined and compared with the results that we conducted from several Matlab scripts. Furthermore, the whole r-GPSR architecture was simulated with the "Behavioral Simulation" process and the "Post Place and Route Simulation" process. Finally, the functionality of our architecture was tested on-board, with the use of a python script and an RS-232 port.

- **FPGA-Atom Board Communication Protocol**: For the verification of this sub-system we designed a python script that sends random values through the RS-232 ports and we expect back, these values inverted. This is done so as to check the development board communication process. Also, in order to check the communication between the ZigBee module, which is programmed with the Base Station Application, and the Atom Board, we created a python

script that receives packets from the air, prints them to the monitor and sends them back to the air, with another ID. For the latter, the ZigBee dongle was utilized.

- **FPGA-based Node**: We program the FPGA to act as a r-GPSR node, with an ID and a location position, and we put it inside an existing network, that consists of several nodes, programmed with the GPSR application. What we expect is to see if this node receives packets, adds nodes to its neighbor list and sends beacon and data packets. All the exchanged packets are captured with the ZigBee dongle.

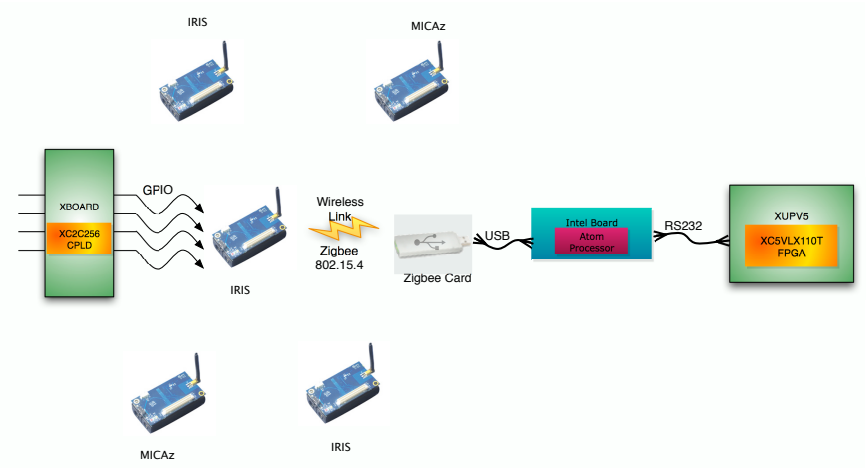

Figure 7.1: TestBed General Scheme

7.1.2 System Testbed

In this section we propose two different testbeds for our new-platforms: The first one includes 2 CPLD-based nodes, an FPGA-based node and several IRIS and MICAz motes. An IRIS mote is used a GPSR Base Station with a random ID and location position. All the other nodes have also random and distinct IDs and location positions. We place all the nodes in different positions inside an area, according to their locations, and start up all the nodes. In order to have a real world experiment, we program the motes and the CPLD-based nodes to sense the

environment (light and temperature) and send these values to the air over the GPSR stack. The FPGA-based node cannot sample any values, since it has no embedded sensors. As a result it acts as a simple packet forwarder. In other words, this node doesn't generate any new data packets.
The second testbed application includes 2 CPLD-based nodes, several IRIS and MICAz motes and a FPGA-Based BaseStation node. All nodes gather temperature and light measurements and try to forward them, through the GPSR stack to the BaseStation, where they are stored into a database and are uploaded to the Internet. A general scheme of this testbed application is presented in figure 7.1.

7.2 Monitoring Tools

In order to test, monitor and measure the wireless link activity, two different options were implemented:

- **ZigBee Dongle:** The messages exchanged among the nodes can be captured by the Integration Wireless Platform Analyzer using a ZigBee dongle (Integration IA-OEMDAUB1-2400 - ZigBee ready, 2.404 - 2.481GHz / IEEE 802.15.4), which is installed on a monitoring PC. These messages are transferred to the PC for further analysis.

- **WEB Interface**: As it was previously mentioned, all the messages received from the FPGA node, both from the ZigBee interface card and the FPGA device, are stored to a database, using the Open Source MySQL Database Management System [101]. These stored packets are then published on the Internet with the help of the open source Apache HTTP web server [102], which is installed on the Atom board.

Chapter 8

Performance Results, Conclusions and Future Work

In this chapter we present the performance results of our previously described implementations. First we present the execution time, energy consumption and max power draw of the CPLD implementation and then the FPGA-based system performance results. Later in this chapter we describe the conclusions of this thesis and the future work that can be done on it, in order to achieve better results and to enhance its scientific contribution.

8.1 Performance Results

8.1.1 CPLD Approach

The system was evaluated based on three major metrics: execution time, energy consumption and maximum power draw. All these are critical parameters in WSNs, since it is certainly desirable to increase the limited processing power of the node while also increasing the life time of the wireless mote by reducing the energy and maximum power consumption.

Our performance results are based on real-world experiments in which a mixed signal oscilloscope has been used in order to take the speed, energy and power measurements. An extra signal has been used in both the software and the hardware implementation of the application in order to measure the execution time; this signal transits to high when the execution of the specific process starts and then toggles back to low, when the process ends. Furthermore, the energy consumption is calculated using the integral of the measured voltage V_m for the measured execution time period Δ_τ. The result is divided with the reference resistance R_{ref}, which is equal to 0.1 Ω in the experimental topology used, in order to calculate the reference current I_{ref}.

Multiplying the I_{ref} with the reference voltage V_{ref} that is equal to 2.7 V for the Micaz Mote and 3.3 V for the CPLD, the overall energy consumption is calculated

based by the following formula:

$$E = I_{ref} V_{ref}, where I_{ref} = \frac{\Sigma_i V_{m,i} \Delta\tau}{R} \tag{8.1}$$

Regarding maximum power consumption, this is calculated by multiplying the reference voltage of the system V_{ref}, with the maximum measured value of the current I_{max}, which is calculated by the division of the measured value for the voltage V_{max}, with the reference resistance R_{ref}. The actual equation used is the one below (8.2).

$$P_{max} = I_{m,max} V_{ref}, where I_{m,max} = \frac{V_{m,max}}{R_{ref}} \tag{8.2}$$

The overall measured values for the specified system are presented in Table 8.1. At

Table 8.1: CPLD Node Approach with GPSR Performance Results

System	**Execution Time(s)**	**Energy Consumption(J)**	**Max Power Draw(W)**
MICAz Mote	3.15E-06	5.96E-07	2.66E-01
MICAz Mote plus CoolRunner-II	9.46E-05	3.29E-05	3.89E-01
MICAz Mote plus CoolRunner-II(overhead)	7.04E-05	1.68E-05	3.72E-01

this point it should be mentioned that there is no point in measuring the execution time, energy consumption and maximum power draw of the CPLD standalone, when this is processing the "Eucidean Distance" algorithm, since it can not stand as a complete system.

In order to have realistic measurements, we repeated the experiments for the previous systems, for 2, 3, 4, 5, 6 and 8 nodes in the network for 100 packets each and we calculated the mean values. We found out, based on our measurements, that there is no significant difference in the execution time, energy consumption or maximum power draw. Also, there is no significant difference, when we use IRIS motes instead of Micaz, since their micro-controllers belong to the same family, so their electrical characteristics remain almost the same.
Regarding the measurement of the overhead of the communication between the

Table 8.2: CPLD with GPSR Resource Utilization

Macrocells	228/256 (90%)
Pterms	640/896 (72%)
Registers	167/256 (66%)
Pins	28/118 (24%)
Function Block Inputs	420/640 (66%)

Table 8.3: Motes with GPSR Resource Utilization

Resource	**IRIS (ATmega1281)**		**MICAz (ATmega128L)**	
	without CPLD	with CPLD	without CPLD	with CPLD
ROM (bytes)	19006	21528	19762	22770
RAM (bytes)	2842	2840	2649	2651

Mote and the CPLD, this is implemented as follows: We programmed the CPLD with a design that doesn't calculate the Euclidean Distance Algorithm, but implements only the communication protocol. The software, with which the mote is programmed remains the same.

Based on our experiments, (and when the CPLD was clocked at a moderate rate of 48MHz), the derived experimental results are not very promising. The execution time when using a CPLD is decreased by 30 times when compared with the execution time of the same algorithm on the micro-controller of the Mote. Moreover, the communication overhead takes about the 75% of the total execution time, due to the large number of bits that have to be transmitted to and from the CPLD. To be more specific, the total number of bits that need to be transmitted are 64 or 8 bytes; as a result, it can be conducted that this specific algorithm is not efficient enough on the CPLD and in combination with the delay that the custom-made cable adds together with the protocol it is not a good tactic to implement this algorithm on the CPLD. Previous work at MHL, have proved that the CPLD can add processing power and

thus reduce the execution time, when the total transmission bits are fewer and when the CPLD executes simpler functions (e.g. encryption, turbo-coding) [103].
Regarding the overall energy consumption for this algorithm is increased by 98.19% whereas more of 48% of this energy is spent for the communication process.
The max power draw for this algorithm when the CPLD is connected is about 31.62% higher compared to the max power draw without a CPLD.
As far as the resource utilization of the CPLD and the Motes concerned, this is presented in tables 8.2 and 8.3 respectively.
Table 8.4 illustrates the performance results of our new platform when the XMesh routing protocol stack is utilized. We measured the execution time, energy consumption and max power draw of the Cost Function that was described in the previous chapter, when this is executed on a mote and on our platform respectively. At this point it should be mentioned, that due to the limited resources of the CPLD device, only the multiplication is executed on the CPLD. The rest of the cost function in executed on the mote. The results derived from real world experiments while the methodology used was the same as for the GPSR Euclidean Distance. As Table 8.5 clearly demonstrates, the execution time remains almost the same in both cases, due to the fact the bottleneck of this function is the division. On the other hand, when using the CPLD for executing the 8-bit multiplication, the overal energy is decreased by 71,5%. Also there is a decrease in in max power draw by almost 50% when the CPLD device is utilized. Comparing these results with those derived from the first system (Table 8.2), we can see that in the latter case the CPLD actually accelerates the performance of the XMesh protocol. This is explained by the fact that the overhead the communication protocol introduces is lower since we have to transfer 4 bytes instead of 8 in the first case. Besides that the multiplication process takes almost the half time to be executed since it operates on 8-bit numbers.
In Table 8.5 the CPLD utilization is presented for the XMesh Cost Function together with the communication protocol. As this table clearly demonstrates, almost half the resources of this devices are utilized. The division process though requires more than these resources so as to be implemented together with the multiplication process.

Table 8.4: CPLD Node Approach with XMesh Performance Results

System	Execution Time(s)	Energy Consumption(J)	Max Power Draw(W)
MICAz Mote	1.25E-04	8.56E-06	1.01
MICAz Mote plus CoolRunner-II	1.23E-04	2.44E-06	5.13E-01

Table 8.5: CPLD with XMesh Resource Utilization

Macrocells	134/256 (53%)
Pterms	338/896 (38%)
Registers	116/256 (46%)
Pins	28/118 (24%)
Function Block Inputs	208/640 (33%)

8.1.2 FPGA Approach

The FPGA system was evaluated on the same metrics: execution time, energy consumption and maximum power draw. All these are metrics of critical importance for WSNs. At this point we compare the GPSR protocol on the FPGA device against its software implementation on the Atom Processor. We do not compare the previous system with this one, since they are used for different purposes: The FPGA is used in cases where there is a great need in processing power such as in WiMax networks. So, it completely pointless to compare the GPSR execution time, energy consumption and maximum power draw on a simple, low power, 8-bit micro controller with a large, state-of-the-art FPGA Device.
In other words, what we examine here is the possibility of replacing the "big" aggregate nodes on a network, that consist only of a general purpose CPU, with our newly introduced reconfigurable system.

Our FPGA performance results are based on real-world experiments in which a mixed signal oscilloscope has been used in order to take the speed, energy and power measurements. An extra signal has been used in both the software and the hardware implementation of the application in order to measure the execution time; this signal transits to high when the execution of the specific process starts and then toggles back to low, when the process ends. Furthermore, the energy consumption is calculated using the integral of the measured voltage V_m for the measured execution time period Δ_τ. The result is divided with the reference resistance R_{ref}, which is equal to 0.1 Ω in the experimental topology used, in order to calculate the reference current I_{ref}.
Multiplying the I_{ref} with the reference voltage V_{ref} that is equal to 3.3 V for the FPGA, the overall energy consumption is calculated based by the following formula:

$$E = I_{ref} V_{ref}, where I_{ref} = \frac{\Sigma_i V_{m,i} \Delta\tau}{R} \tag{8.3}$$

Regarding maximum power consumption, this is calculated by multiplying the reference voltage of the system V_{ref}, with the maximum measured value of the current I_{max}, which is calculated by the division of the measured value for the voltage V_{max}, with the reference resistance R_{ref}. The actual equation used is the one below (8.4).

$$P_{max} = I_{m,max} V_{ref}, where I_{m,max} = \frac{V_{m,max}}{R_{ref}} \tag{8.4}$$

The FPGA board used for our measurements is the **Digilent XUPV2P board, with a Virtex 2 Pro XC2VP30 integrated FPGA device**, since the XUPV5 board used for our implementation can not provide the appropriate input and output pins for this kind of measurements. We cannot isolate the voltage regulator, and thus we cannot power up the FPGA device from an external Power Generator. In order to measure the execution time of the GPSR software on the Atom Processor, we used the Intel Vtune Performance analyser tool 9.1 [104] installed on Linux Xubuntu 8.10, whereas for the power draw measurement we used the Linux PowerTop tool [105]. PowerTop is a tool that measures the percentage of power states at which a processor exists, when executing a software. We couldn't used the previously described methodology, in order to measure the execution time, energy and power draw of the Intel Atom Board, since we couldn't isolate the voltage regulator, which drives the current to the processor. This is the only technique that is used in bibliography, in order to measure the power consumption of this kind

Table 8.6: FPGA vs CPU Performance Results **(XUPV2P)**

System	Execution Time(s)	Energy Consumption(J)	Max Power Draw(W)
Reconfigurable Node	1.36E-04	3.50E-06	1.40E-01
Reconfigurable BaseStation	1.52E-05	2.18E-07	3.54E-02
ATOM Processor(Node+BS)	4.28E-03	1.09E-02	2.54

of boards. Regarding the power states of the Atom Processor, these are described in detail in [106] data-sheet. To summarize, the Atom 330 Family processor has 2 states : C0 and C1. The C0 state is the normal operating state for threads in the processor. C1 is a low power state entered when a thread executes a halt or wait instructions. When the Atom processor enters in C0 state, the average power draw is 8 W, whereas when it enters in C1 state the average power draw is 2 W. These are the only experimental values the manufacturer provides, without giving details for this specific the experimental topology. Also, there is no way to measure the average energy consumption, since there is not available tool for this kind of measurement. In order our measurements on the FPGA to keep up with the measurements on the Atom processor, the same experiments were carried out: The same number of packets were exchanged among the nodes of a network that was set up, with the same beacon delay and the same topology (locations) of the nodes inside the network.
The overall measured values for the specified system are presented in Table 8.6, while the FPGA resource utilization results are presented in Tables 8.8 and in 8.7 for the Virtex 2 Pro and for the Virtex 5 device respectively, for both Base Station and Node applications. All the utilization results are those that the ISE tool generated after the "Place and Route" process (at the Place and Route Report). In the following tables we include also the power results that the Xilinx XPower Analyzer tool generated [92]. These results are presented here, in order for consistency to exist with the measurements on the Intel Atom Board. Those results were measured with the use of a software tool (PowerTop) and as a consequence, it is necessary, to present the FPGA power consumption values that were derived by a software tool (Xilinx XPower Analyzer) too. Regarding the FPGA performance results, we measured the execution time, the energy consumption and the max power draw for 1000 packets (beacon+data). The measurements were taken by sets of 100 packets (10

Table 8.7: Virtex 5 Resource Utilization

Resource	Base Station	Node
DSP48Es	-	25 out of 64 (39%)
RAMB18x2s	1 out of 148 (1%)	1 out of 148 (1%)
RAMB36 EXPs	1 out of 148 (1%)	1 out of 148 (1%)
Slice Registers	1608 out of 69120 (2%)	14910 out of 69120 (21%)
Slice LUTS	1630 out of 69120 (2%)	16485 out of 69120 (23%)
Slice LUT-Flip Flop pairs	2166 out of 69120 (3%)	20433 out of 69120 (29%)
Frequency (MHz)	165.80	56.50
Power Cons. (W) XPower Measurement	0.98	1.07

sets of 100 packets each). The packets were random, and generated by the motes. On the other hand, for the Atom measurements, we used the same node's locations and took the measurements for 1000 packets: using the VTune for the execution time and PowerTop for the power draw respectively.

The comparison of the performance results in table 8.6, shows a speedup in execution time from the FPGA side by a factor of 31 and a decrease in power draw by almost 95%.

As far as the FPGA utilization results concerned, the critical path of our design is the module that implements the face change algorithm. It consists of many complex logic that delays the execution of the whole system and thus the frequency achieved by the tools is not very high compared to the maximum frequency that can be achieved on these FPGA devices (according to the manufacturers).

At this point it should be also mentioned that the XPower Analyser values are presented here for consistency purposes with the results derived from the Linux PowerTop tool. As it is clearly demonstrated by the derived values, the XUPV2P results are very similar (same order of magnitude) to the results taken by the experimental measurements with the oscilloscope. Their difference relies on the facts that, firstly the XPower Analyzer makes only an estimation based on the design. before this is downloaded on the board and secondly by the fact this tools estimates the worst power consumption that the board may have during real time operation.

Table 8.8: Virtex 2 Pro Resource Utilization

Resource	**Base Station**	**Node**
MULT18x18s	-	18 out of 136 (13%)
RAMB16s	3 out of 136 (2%)	3 out of 136 (2%)
Slices	1381 out of 13696 (10%)	12459 out of 13696 (90%)
Frequency (MHz)	122.69	41.16
Power Cons. (W) XPower Measurement	0.12	0.23

In Table 8.9 the throughputs of the Reconfigurable Sensor Nodes approaches are presented. The length of the beacon and data packets is 120 bits and 272 respectively. This value is necessary in order to estimate the bandwidth of our system in the case which it will be used as part of a 802.11 Network (WiFi or WiMax). Measuring the bandwidth at the case of ZigBee is useless, since not only its a limited-bandwidth protocol but also it was used as the most convenient one during the design and implementation of our systems. At this point it should also be mentioned that our system uses as I/O device the RS232 protocol, for the same purpose. If we have embedded a USB ZigBee Card on the FPGA or if we have chosen another I/O protocol the usage of the Atom Board would have been unnecessary.

Table 8.9: Throughput (MPackets/s) FPGA Approach

Packet Type	**XUPV2P**		**XUP5V**	
	BaseStation (122.69 MHz)	Node (41.16 MHz)	BaseStation (165.80 MHz)	Node (56.50 MHz)
Beacon (120 bits)	5.88	1.88	7.69	2.56
Data (272 bits)	-	0.56	-	0.75

In table 8.10 the maximum throughput values in different transmission protocols are presented. These values are calculated based on the maximum packet sizes that each of these protocols can support. As it is clearly demonstrated, in Wifi, the best throughput can be achieved.

Table 8.10: Maximum Throughput in different Protocols

Protocol/ Max. Packet Size	Max. Throughput Beacon Packet (MB/second)	Max. Throughput Data Packet (MB/second)
ZigBee/128 Bytes	327.68	96.00
Ethernet/518 Bytes	1326.08	388.50
WiFi/2312 Bytes	5918.72	1734.00

8.2 Conclusions

In this thesis we presented a new approach to the design of routing protocols with the use of reconfigurable devices. We introduced two different approaches: the first one consists of common used WSN motes and state-of-the-art low-power CPLD devices while the second one consists of a large FPGA device and a general purpose CPU. We designed the platforms, together with the appropriate hardware and software and we evaluated them with the design of 2 different reconfigurable architectures of the GPSR and XMesh protocols respectively.

Regarding the first approach, we implemented in hardware commonly used by the routing protocols functions and we connected the CPLD device with Xbow's motes. We designed and implemented both in software and hardware a communication protocol for these two devices, and we set up a WSN to validate this platform's performance results, based on execution time, energy consumption and maximum power draw attributes. The results that came out from our experimental measurements were not very promising in the case of GPSR, since no speedup or energy saving was achieved, due to the large overhead that is introduced by the communication protocol, and also due to the fact that the mote micro-controller is very capable of executing arithmetic operations (more efficient than a custom approach). In the case of the XMesh, where the arithmetic operations are smaller, there was a slight speedup in execution time and we achieved energy consumption and a reduction in max power draw.

The results that derived form the second approach were very promising (compared to the first one). We designed and implemented the whole GPSR protocol on a state-of-the-art FPGA device. We also implemented all the necessary software suite on the an Atom processor, that is used for I/O purposes with the FPGA from the one side and with the air from the other. The speed-up achieved is about 31 times

higher compared to the GPSR software that runs on an Atom Processor, while the max power draw is decreased at about 95%. Besides that, a whole new platform was introduced, on which many WSN applications can be implemented.

8.3 Future Work

Many things can be done in order to improve our systems. First of all, in order to improve the energy consumption and power draw of the CPLD, a more-energy efficient communication protocol can be designed for the CPLD-Mote platform. Besides that, another part of the GPSR can be designed on it, or at least another more CPLD-appropriate application, such as encryption or Turbo-coding.
Also, the custom made cable can be replaced together with all the development areas of the boards, since they waste a lot of energy. Instead of these, a custom board can be designed which contains only a CPLD device, the Mote's micro-controller and the ZigBee module.
Except for that, a larger CPLD device can be used, in order to explore the energy consumptions when a bigger device is utilized and consequently another application that uses all its resources.
Regarding the FPGA-based architecture, we can enhance our protocol functionality and security by adding some trust metrics, as these referred in [107].
Also we can embed sensors on our FPGA so as our node to have the ability to gather environmental measurements and send them over the network.
Finally, we can replace the RS-232 connection of the Intel Atom Board, with Gigabit Ethernet, in order to succeed higher transfer rates between the FPGA and the Atom processor and thus enabling the use of this protocol to other network types, such as WiMax.

Appendix

RS-232 hardware module

As referred in implementation chapter, serial port is used by the base station for sending/receiving data to/from the FPGA device. For this purpose, a RS-232 hardware module is implemented in order to be used as input/output interface from the side of the reconfigurable device. An abstract block diagram of this module is illustrated in Figure 1.

The components that are used for communicating with the external port are the

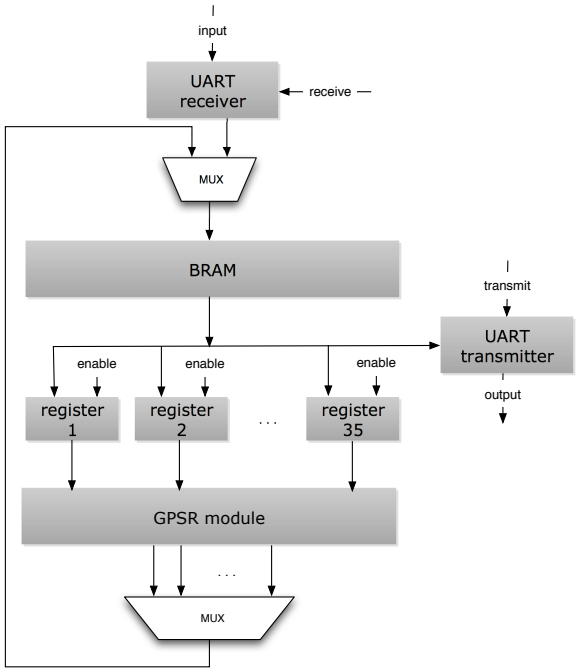

Figure 1: RS-232 Block Diagram

UART receiver and transmitter; the first one is used for receiving data from the port and the other one for transmitting data to it. Another component used for the same purpose is called "baud" and is a clock divider used for adjusting the system clock with the UART one.

RS-232 module also consists of two multiplexers (a 2-1 and a 35-1 with 8-bit inputs/output) and 35 8-bit registers (34 are the bytes of the GPSR data packet plus one indicating the type of the packet). Also a BRAM, generated by Xilinx Core Generator and organized by 1024 memory location of 8 bit each, is used in order to store the input and output data.

Further more we designed a controller which arbitrates the functionality of the

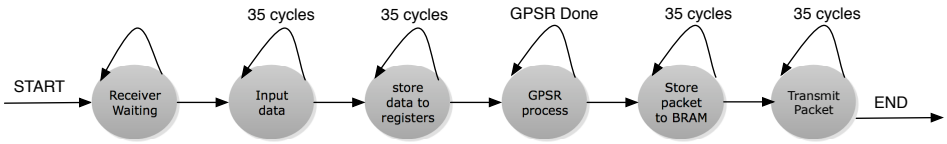

Figure 2: UART Controller

previously described RS-232 module. According to the the timing diagram defined, the UART receiver is initially in waiting state. When an interrupt occurs, the receiver is enabled. This data is received in quantities of 8 bits each one. This process is repeated 35 times, until the whole GPSR packet fields are succesfully received and stored into the BRAM. The first multiplexer is set to select the output of the UART receiver in order to store the input data. The following 35 cycles are spent in order to read this input data byte-by-byte (reading each position of the memory) and store it to the 35 byte-registers, the outputs of which are connected to the input of the GPSR module. This module then outputs a new packet (beacon or data), and stores it to the RS-232 BRAM. Finally, the output data are read from the BRAM and sent to the UART transmitter during the last 35 clock cycles in order to send it to the serial port enabling its transmit interrupt. The gpsr controller is presented in figire 2.

Bibliography

[1] J. N. Al-Karaki and A. E. Kamal. Routing techniques in Wireless sensor Networks: a survey. *IEEE Wireless Communications*, 11(6):6–28, 2004.

[2] Nirupama Bulusu, John Heidemann, and Deborah Estrin. GPS-less Low Cost Outdoor Localization for very small devices, 2000.

[3] Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in Ad-Hoc networks of sensors. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 166–179, New York, NY, USA, 2001. ACM.

[4] Lan F. Akyildiz, Welljan Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on Sensor Networks, 2002.

[5] Sameer Tilak, Nael B. Abu-Ghazaleh, and Wendi Heinzelman. A taxonomy of Wireless micro-sensor Network models. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(2):28–36, 2002.

[6] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society.

[7] Fan Ye, Haiyun Luo, Jerry Cheng, Songwu Lu, and Lixia Zhang. A two-tier data dissemination model for large-scale wireless sensor networks. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 148–159, New York, NY, USA, 2002. ACM.

[8] Fan Ye, Alvin Chen, Songwu Lu, Lixia Zhang, and Fan Ye Alvin Chen. A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks. pages 304–309, 2001.

[9] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, New York, NY, USA, 1999. ACM.

[10] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wirel. Netw.*, 8(2/3):169–185, 2002.

[11] Hedetniemi, Hedetniemi, and Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *NETWORKS: Networks: An International Journal*, 18, 1988.

[12] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM Press.

[13] David Braginsky and Deborah Estrin. Rumor Routing Algorithm for Sensor Networks. pages 22–31, 2002.

[14] Curt Schurgers Mani and Mani B. Srivastava. Energy Efficient Routing in Wireless Sensor Networks. In *in the MILCOM Proceedings on Communications for Network-Centric Operations: Creating the Information Force*, pages 357–361, 2001.

[15] Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable Information-Driven Sensor Querying and Routing for ad hoc Heterogeneous Sensor Networks, 2002.

[16] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[17] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. The ACQUIRE Mechanism for Efficient Querying in Sensor Networks. In *In IEEE International Workshop on Sensor Network Protocols and Applications (SNPA03*, pages 149–155, 2003.

[18] Rahul C. Shah and Jan M. Rabaey. Energy aware routing for low energy Ad-Hoc Sensor Networks, 2002.

[19] Sergio D. Servetto and Cornell Univerisity. Constrained random walks on random graphs: Routing algorithms for large scale wireless sensor networks. pages 12–21, 2002.

[20] Corresponding Author Cauligi, Stephanie Lindsey, Cauligi S. Raghavendra, and Cauligi S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems stephanie lindsey cauligi s. raghavendra.

[21] Arati Manjeshwar and Dharma P. Agrawal. TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 189, Washington, DC, USA, 2001. IEEE Computer Society.

[22] Arati Manjeshwar and Dharma P. Agrawal. APTEEN: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 48, Washington, DC, USA, 2002. IEEE Computer Society.

[23] Volkan Rodoplu and Teresa H. Meng. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 17:1333–1344, 1998.

[24] Qing Fang, Feng Zhao, and Leonidas Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *In Proceedings of the 4th ACM International Symposium on Mobile ad hoc networking and computing*, pages 165–176. ACM Press, 2003.

[25] Qun Li, Javed Aslam, and Daniela Rus. Hierarchical power-aware routing in sensor networks. In *In Proceedings of the DIMACS Workshop on Pervasive Networking*, 2001.

[26] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wirel. Netw.*, 8(5):481–494, 2002.

[27] Jae-Hwan Chang and Leandros Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(4):609–619, 2004.

[28] Stefan Dulman, Tim Nieberg, Jian Wu, and Paul Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks, 2003.

[29] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(4):11–25, 2001.

[30] Katayoun Sohrabi, Jay Gao, Vishal Ailawadhi, and Gregory J Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7:16–27, 2000.

[31] N. Lewis, N. Foukia, and D. G. Govan. Using trust for key distribution and route selection in wireless sensor networks. In *In IEEE Globecom Workshops November 2007*, 2007.

[32] G. Theodorakopoulos and J. S. Baras. On trust models and trust evaluation metrics for ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 24(2):318–328, 2006.

[33] Zhaoyu Liu Joy and R.A. A.W. Thompson. A Dynamic Trust Model for Mobile Ad Hoc Networks. In *FTDCS '04: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pages 80–85, Washington, DC, USA, 2004. IEEE Computer Society.

[34] Ismat K. Maarouf and A. R. Naseer. WSNodeRater - an Optimized Reputation System Framework for security aware energy efficient geographic routing in WSNs. *Computer Systems and Applications, ACS/IEEE International Conference on*, 0:258–265, 2007.

[35] Asad Amir Pirzada and Chris McDonald. Performance comparison of trust-based reactive routing protocols. *IEEE Transactions on Mobile Computing*, 5(6):695–710, 2006. Member-Datta, Amitava.

[36] Nael Abu-Ghazaleh, Kyoung-Don Kang, and Ke Liu. Towards resilient geographic routing in wsns. In *Q2SWinet '05: Proceedings of the 1st ACM*

international workshop on Quality of service & security in wireless and mobile networks, pages 71–78, New York, NY, USA, 2005. ACM.

[37] Sapon Tanachaiwiwat, Pinalkumar Dave, Rohan Bhindwale, and Ahmed Helmy. Location-centric isolation of misbehavior and trust routing in energy-constrained sensor networks. In *IEEE Workshop on Energy-Efficient Wireless Communications and Networks (EWCN), in conjunction with IEEE IPCCC*, pages 14–17, 2004.

[38] Riaz Ahmed Shaikh, Hassan Jameel, Sungyoung Lee, Saeed Rajput, and Young Jae Song. Trust management problem in distributed wireless sensor networks. In *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 411–414, Washington, DC, USA, 2006. IEEE Computer Society.

[39] A. A. Pirzada and C. McDonald. Trust establishment in pure ad-hoc networks. *Wirel. Pers. Commun.*, 37(1-2):139–168, 2006.

[40] Garth V. Crosby, Niki Pissinou, and James Gadze. A framework for trust-based cluster head election in wireless sensor networks. In *DSSNS '06: Proceedings of the Second IEEE Workshop on Dependability and Security in Sensor Networks and Systems*, pages 13–22, Washington, DC, USA, 2006. IEEE Computer Society.

[41] Matthew J. Probst and Sneha Kumar Kasera. Statistical trust establishment in wireless sensor networks. In *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems*, pages 1–8, Washington, DC, USA, 2007. IEEE Computer Society.

[42] Dimitrios Lymberopoulos, Nissanka B. Priyantha, and Feng Zhao. mPlatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 128–137, New York, NY, USA, 2007. ACM.

[43] Mikko Kohvakka, Tero Arpinen, Marko Hännikäinen, and Timo D. Hämäläinen. High-performance multi-radio wsn platform. In *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 95–97, New York, NY, USA, 2006. ACM.

[44] Mikko Kohvakka. Atific helicopter high performance multi-radio wsn platform.

[45] Altera. http://www.altera.com/. Last accessed: 29-06-2009.

[46] Altera. *Nios* II *Processor Reference Handbook*, March 2009.

[47] Altera. *Cyclone* FPGA *Datasheet*, March 2008.

[48] Nordic Semiconductor. http://www.nordicsemi.com/. Last accessed: 29-06-2009.

[49] Inc All Rights Reserved. RS232 library 1.0 1.0.

[50] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, New York, NY, USA, 2000. ACM.

[51] Jian Chen, Yong Guan, and Udo Pooch. Customizing a geographical routing protocol for wireless sensor networks. In *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, pages 586–591, Washington, DC, USA, 2005. IEEE Computer Society.

[52] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 217–230, Berkeley, CA, USA, 2005. USENIX Association.

[53] Matthias Witt and Volker Turau. The impact of location errors on geographic routing in sensor networks. In *ICCGI '06: Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, page 76, Washington, DC, USA, 2006. IEEE Computer Society.

[54] William T. Zaumen and J. J. Garcia-Luna Aceves. Dynamics of distributed shortest-path routing algorithms. *SIGCOMM Comput. Commun. Rev.*, 21(4):31–42, 1991.

[55] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM.

[56] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[57] Charles Perkins and Elizabeth Royer. Ad-hoc on-demand distance vector routing. In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1997.

[58] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *DIALM '99: Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 23–29, New York, NY, USA, 1999. ACM.

[59] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Trans. Netw.*, 2(2):122–136, 1994.

[60] Ruben K. Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, September 1969.

[61] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980.

[62] N. Abramson. The aloha systemanother alternative for computer communications. In *Proceedings of Fall Joint Computer Conference, AFIPS Conference*, pages 37+, 1970.

[63] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[64] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard 802.11, June 1999.

[65] Phil Karn. MACA: a new channel access method for packet radio. In *Computer Networking Conference*, volume 9th, pages 134–140, 1990.

[66] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: a media access protocol for wireless LAN's. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 212–225, New York, NY, USA, 1994. ACM.

[67] Awissenet European Project. http://www.awissenet.eu/. Last accessed: 29-06-2009.

[68] Asad Amir Pirzada and Chris McDonald. Trusted greedy perimeter stateless routing. In *15th IEEE International Conference on Networks, 2007 (ICON 2007)*, pages 206–211, 2007.

[69] *TinyOs Online Tutorial*. http://www.tinyos.net, September 2008. Last accessed: 29-06-2009.

[70] Kim Young Jin and Godivan Ramesh. GPSR *Detailed-level Design Specification*. University of Southern California, July 2003.

[71] Kim Young Jin and Godivan Ramesh. GPSR *User Specification*. University of Southern California, July 2003.

[72] Red Hat Linux. http://www.redhat.com. Last accessed: 29-06-2009.

[73] Ubuntu Linux. http://www.ubuntu.com. Last accessed: 29-06-2009.

[74] Crossbow. http://www.xbow.com. Last accessed: 29-06-2009.

[75] HandHelds. http://www.handhelds.org/geeklog/index.php. Last accessed: 29-06-2009.

[76] Kim Young Jin and Godivan Ramesh. GPSR *Design Specification*. University of Southern California, May 2004.

[77] Crossbow. MPR-MIB *Users Manual*, June 2007. Revision A.

[78] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM.

[79] Crossbow Technologies. *XMesh Users's Manual*, March 2007. Revision C.

[80] A. Teo, G. Singh, and J.C. McEachen. Evaluation of the xmesh routing protocol in wireless sensor networks. *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, 2:113–117, Aug. 2006.

[81] Umberto Malesci and Samuel Madden. A measurement-based analysis of the interaction between network layers in tinyos. In *EWSN*, 2006.

[82] P. Buonadonna, D. Gay, J.M. Hellerstein, W. Hong, and S. Madden. TASK: sensor network in a box. *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 133–144, Jan.-2 Feb. 2005.

[83] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, New York, NY, USA, 2003. ACM.

[84] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM.

[85] Mark D. Yarvis, W. Steven Conner, Lakshman Krishnamurthy, and Alan Mainwaring. Intel labs.

[86] Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers, and Robert Morris. Performance of multihop wireless networks: shortest path is not enough. *SIGCOMM Comput. Commun. Rev.*, 33(1):83–88, 2003.

[87] Xilinx. *Xilinx Core Generator*, June 2008.

[88] Xilinx. *Divider v2.0 Product Specification*, June 2008.

[89] Xilinx. *CoolRunner-II* CPLD F*amily*. Xilinx, v3.1 edition, September 2008.

[90] Digilent. *X*B*oard Reference Manual*, January 2007.

[91] HDL Analysis and Standardization Group. http://www.eda.org/vhdl-200x/. Last accessed: 29-06-2009.

[92] Xilinx ISE 10.1. http://www.xilinx.com/ise/. Last accessed: 29-06-2009.

[93] ModelSim Simulator. http://www.model.com/. Last accessed: 29-06-2009.

[94] Crossbow. MDA *Users Manual*, June 2007. Revision A.

[95] Xilinx. *Virtex-5 Family Overview*. Xilinx, v5.0 edition, February 2009.

[96] Xilinx. ML505/ML506/ML507 *Evaluation Platform*. Xilinx, v3.1 edition, November 2008.

[97] Intel. I*ntel* D*esktop* B*oard* D945GCLF2*, Technical Product Specifiaction*. Intel, December 2008.

[98] Xubuntu Linux. http://www.xubuntu.com. Last accessed: 29-06-2009.

[99] Python Online. http://www.python.org. Last accessed: 29-06-2009.

[100] TinyOS Serial Forwarder tool. http://docs.tinyos.net/index.php/. Last accessed: 29-06-2009.

[101] MySQL DBMS. http://www.mysql.com/. Last accessed: 29-06-2009.

[102] Apache Server. http://www.apache.org/. Last accessed: 29-06-2009.

[103] Mplemenos Georgios, Papadopoulos Konstantinos, Brokalakis Andreas, Grigorios Chrysos, Sotiriades Euripides, and Papaefstathiou Ioannis. RESENSE: Reconfigurable WSN Nodes. In *WISIG: Wireless Sensing Showcase 2009, London, UK*, 2007.

[104] Intel VTune Performance Analyzer. http://software.intel.com/en-us/intel-vtune/. Last accessed: 29-06-2009.

[105] PowerTop. http://www.lesswatts.org/projects/powertop/. Last accessed: 29-06-2009.

[106] Intel. *Intel* A*tom* P*rocessor 330 Systems Datasheet*, February 2009. Revision 002.

[107] Pirzada A. and McDonald C. Trusted greedy perimeter stateless routing. In *15th IEEE International Conference on Netowrks*, pages 206–211, 2007.

[108] C. S. Raghavendra. *Wireless Sensor Networks*. Springer, July 2005.

[109] Riesgo Teresa Portilla Jorge and Angel de Castro. A reconfigurable FPGA-based architecture for modular nodes in wireless sensor networks. In *3rd Southern Conference on Programmable Logic,2007, SPL '07*, pages 203–206, 2007.

[110] Chee yee Chong, Ieee, Srikanta P. Kumar, and Senior Member. Sensor networks: evolution, opportunities, and challenges. In *Proceedings of the IEEE*, pages 1247–1256, 2003.

[111] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In ASPLOS-IX*: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104, New York, NY, USA, 2000. ACM.

[112] *Nes*C *Tutorial Online*. http://en.wikipedia.org/wiki/NesC. Last accessed: 29-06-2009.

[113] Atmel. AT*mega128L Datasheet*, September 2003. Revision I.