UNIVERSITAT POLITÈCNICA DE CATALUNYA
(TECHNICAL UNIVERSITY OF CATALONIA)
DEPARTAMENT DE LLENGUATGES I SISTEMES INFORMÀTICS
(SOFTWARE DEPARTMENT)
MASTER IN COMPUTING

MASTER THESIS

DEVELOPMENT AND EXPERIMENTAL TESTING AND COMPARISON OF TOPOLOGY-CONTROL ALGORITHMS IN SENSOR NETWORKS

STUDENT: JUAN ANTONIO FARRÉ BASURTE
DIRECTOR: DR. JORDI PETIT I SILVESTRE

DATE: SEPTEMBER 2010

Acknowledgements

Thanks to my mother and brother for their love and support,
and to my uncle and aunt for their invaluable help.

Thanks to the people of Coalesenses GmbH
for their rapid, kind and high-quality support.

Thanks to Dr. Maria J. Blesa
for her advice and support.

Thanks to Dr. Josep Diaz
for giving me the opportunity of working with his team.

Thanks to the WISEBED project
of the ALBCOM research group for their financial support.

And, finally, thanks to my master thesis director
Dr. Jordi Petit for his tutelage and dedication.

Contents

Chapter 1

Introduction

In this Master Thesis we perform an experimental evaluation of topology-control protocols real in wireless sensor networks.

Topology control, that will be described in detail later in this document, is considered a fundamental technique to reduce energy consumption and radio interference. But, to the best of our knowledge, it has only been tested using simulators and there are no evaluations of topology-control protocols in real environments. With this work we intend to fill this gap.

1.1 Simulation vs. real experimentation

Simulation has some important advantages that many times make of it the method of choice to test the results of theoretical research:

- It is fast. Both development and deployment are faster in simulators than in real hardware platforms.
- It is cheap. A single cheap desktop PC is usually enough.
- Allows to test in arbitrary large networks. With a simulator, one can easily simulate networks with thousands of nodes.
- Allows to experiment with very different networks, both in size, topology, and other characteristics.
- Development of algorithms is easier, as it is not necessary to deal with the important constraints that real devices impose, both in the available API and in device memory and speed.

For these reasons, simulation is a very important first step in experimentation. But it is not enough, for the following reasons:

- It is not possible to simulate all the heterogeneous characteristics of real environments that influence communications.
- It is not always possible to test the algorithm behaviour in presence of unreliable communications.

- Simulated nodes do not have restrictions of memory capacity or processing power that could make an algorithm useless in practice.
- There is information that cannot be retrieved from a simulated node because it just does not contain it. For example, it is not possible to measure energy consumption, as simulated nodes do not consume energy and have no battery.
- Even the most careful, complete and exact simulation of a node and a network is never guaranteed to be perfect and, hence, a simulation can never be considered as evidence of the behaviour in real environments.

1.2 Objectives of the thesis

The objectives of this master thesis are the following:

- Development of a selection of topology-control protocols for the Wiselib, and testing them in UPC testbed.
- Development of a routing protocol that relies on the topology generated by a topology-control protocol to select routes.
- Design of experiments to evaluate and compare the protocols in real environments.
- Development of the software needed to perform these experiments.
- Execution of the experiments.
- Analysis of the results.

1.3 Organization of the document

This document is organized in chapters as follows:

- In chapter 2, we give an overview of topology control and of the most important topology-control protocols.
- In chapter 3, we introduce the Wiselib and give details of the implementation of a selection of topology-control protocols in the Wiselib.
- In chapter 4, we perform a theoretical discussion of the different protocols, explain the design of the experiments and development of needed software, and analyse the results of these experiments.
- Finally, in chapter 5, we elaborate the conclusions of this work and suggest ideas for future work.

Chapter 2

Topology control

In this chapter we describe topology control and a selection of well-known topology-control protocols.

2.1 An overview of topology control

One of the most important concerns regarding wireless sensor networks is energy consumption. These devices are usually battery powered and energy capacity is very limited, determining network lifetime. Different techniques arise to deal with this problem, with different approaches. One way to save energy is to reduce the duty cycle of the nodes, letting them disconnect the radio and enter in a power-saving mode while inactive. Another source of energy consumption is radio message transmission. Topology-control protocols try to reduce overall consumption by decreasing the amount of energy needed for wireless communications.

The communication topology of a wireless sensor network can be represented by a graph $G = (V, E)$, where each vertex $v \in V$ represents a sensor node and each edge $(v_1, v_2) \in E$ indicates that nodes v_1 and v_2 can communicate (at maximum power). In the general case, communication is not always bidirectional and, hence, the graph can be directed.

We know that transmission power needed to communicate is proportional, in ideal conditions, to the square of the distance between nodes. In real conditions, obstacles between nodes and other characteristics of the environment further limit communication. In the general case, transmission power needed for two nodes to communicate can be approximated to be proportional to the distance raised to a power $\alpha \geq 2$. This implies that often a multihop transmission through one or more intermediate nodes can be more power efficient than direct communication.

The general objective of a topology-control protocol is finding a subgraph $G' = (V, E')$ of the basic topology graph G, where $E' \subset E$ is such that needed transmission power is minimized while preserving connectivity. Reducing transmission power decreases radio signal range. This has the additional effect of reducing the probability of collision between different transmissions, thus decreasing radio interference and increasing network capacity.

The main goals of a topology-control protocol are:

- Minimizing the power spent by the network in radio message transmission.

- Generating a symmetric topology, that greatly simplifies routing protocols.
- Reducing node degree (number of other nodes that a given node directly communicates with).

The protocol should preferably be distributed and localized (every node has knowledge of the information in its immediate neighbourhood).

For detailed information about topology control, please refere to [15] and [16].

2.2 Topology-control protocols

In this section we provide a brief description of some well-known topology-control protocols. In chapter 4, we will compare these algorithms both at a theoretical and at a practical level.

2.2.1 R&M protocol

The R&M algorithm [13] builds a topology that is optimized for the all-to-one communication pattern, where one of the network nodes is designated as the master node, and all the other nodes send messages to the master. This is achieved by means of finding the optimal solution to the problem of finding the reverse spanning tree of minimum energy cost rooted at the chosen master node.

2.2.2 LMST protocol

The Local Minumim Spanning Tree algorithm (LMST) proposed in [11] builds an approximation of a minimum spanning tree based on positional estimates, which is a good approach for getting minimal energy consumption. To do so, every node in the network locally generates a minimum spanning tree of its visible neighbourhood and then chooses its closest nodes as its own neighbours in the topology.

2.2.3 FLSS protocol

The Fault-tolerant Local Spanning Subgraph (FLSS) protocol [8] is a variation of the LMST protocol aimed at improving the fault tolerance of the constructed topology. In particular, the design goal is to build an energy-efficient topology that preserves k-connectivity, where k is a small constant (typically, 2-3).

2.2.4 KNeigh protocol

Given a network G, the k-neighbors graph G_k is the directed graph obtained connecting each node to its k closest neighbors. The symmetric k-neighbors subgraph G_k^- is obtained by removing all the unidirectional links in G_k. The KNeigh protocol [3] is a distributed implementation of the computation of G_k^- based on distance estimates.

2.2.5 XTC protocol

In the XTC algorithm [19], every node generates an order on the nodes in its immediate neighbourhood, based on decreasing link quality, and broadcasts it. Then it sequentially considers nodes in its generated order. Informally speaking, a node u selects a

neighbour node v to be included in its generated topology if u does not have another neighbour w with better link quality that can be reached more easily from v that u itself.

2.2.6 CBTC protocol

The CBTC algorithm [7, 12] is based on the following idea: set the transmit power level of node u to the minimum value $P_{u,\rho}$ such that u can reach at least one node in every cone of angle ρ centered at u. In other words, a node must retain connections to at least one neighbor in "every direction", where parameter ρ determines the granularity of what is meant by "every direction".

Chapter 3

Wiselib

In this chapter, we descibe the Wiselib, the topology-control concept created for the implementation of different topology-control protocols, and the implementation details of the corresponding models.

3.1 An overview of the Wiselib

Wiselib [2, 20] is a generic algorithm library for heterogeneous wireless sensor nodes. This library is under current development within the WISEBED project [4].

Wiselib will eventually cover a large number of algorithmic topics, including routing, clustering, time synchronization, localization, data dissemination, target tracking and topology control. The library already contains some algorithms in some of these categories, and is currently being enriched by WISEBED partners. The goal is to develop a library of algorithms for heterogeneous WSN's that is on par with some well-known centralized algorithm libraries existing nowadays (such as LEDA, CGAL or BOOST). This aim has strong requirements:

- Wiselib must run on all sensor nodes by WISEBED partners and should easily be ported to additional devices.
- Its algorithms should utilize the capabilities of the device they are compiled for.
- Its memory overhead should be as low as possible compared to a native implementation for a specific device.
- Its algorithms should be highly efficient considering the capabilities of the devices.
- Its algorithm implementations should not explicitly deal with platform-specific dependencies.

Fig. 3.1 depicts how Wiselib library is connected with other components of a WSN system. The library can be used by any user application that needs any of the algorithms implemented in it or, alternatively, that needs to implement a new algorithm using the components that the library offers.

The algorithms included in Wiselib itself are organized in topics according to their functionality. In order to abstract the algorithms from the particularities of the physical platform of sensors and the operating system administrating that platform, a set of connectors exists that defines and fixes an interface for interacting with them. A connector is also defined to interact with wireless-sensor-network simulators as, for

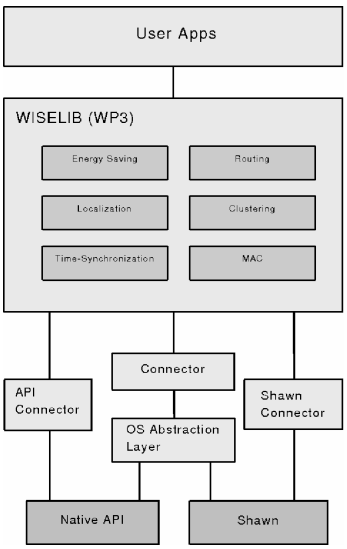

Figure 3.1: Wiselib architecture and external interface [20]

example, Shawn [14]. Those connectors are defined in a way that the same algorithm can be run on a physical platform or on the simulator. Details on how this is achieved are provided in the following.

C++ templates The programming language chosen to implement and use Wiselib is C++ [17]. This decision allows the use of modern programming techniques via object-oriented (OO) design, and provides mostly type-safe development. Moreover, the language offers interesting features such as const-correctness and templates [18, 1]. Wiselib massively uses templates, especially to develop very efficient and flexible applications. The basic functionality of templates is to allow the use of generic code that is resolved by the compiler when specific types are given. Thereby, only the code that is really needed is generated, and methods and parameters as template parameter can be directly accessed. No virtual inheritance is used at all, to avoid the vtables that are necessary to implement virtual function calls. Instead, OO concepts are implemented using template specializations. Using templates and member templates provides the Wiselib with several advantages, such as early binding, inline optimizations, code pruning, extensibility, and a layered structure.

Concepts and models The Wiselib library is not an ordinary object-oriented design with some interfaces and abstract classes. Instead, it uses an efficient generic programming approach that makes extensive use of templates. Thus, an appropriate structure for the description of the several algorithms and components is needed. We heavily employ two generic programming principles (see Fig. 3.2):

Concepts. A concept is a detailed description of the requirements for the basic common functionalities of a class of algorithms, that is, an informal prototype for it. For example, later we present the topology-control concept. Concepts do not contain any source code but, instead, are part of the Wiselib documentation. The

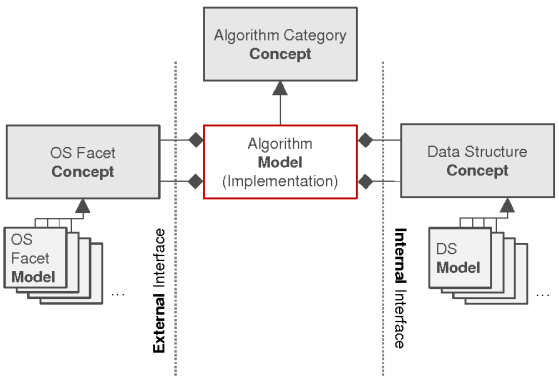

Figure 3.2: Wiselib software architecture [2]

idea is to provide a complete documentation, in which the concepts and their interrelationships are described, and from which one is able to produce a valid implementation. Interestingly enough, concept inheritance is allowed in Wiselib.

Models. A model is a specific implementation of a concept. Any algorithm model implements one or multiple concepts. For example, later we present in detail the concept of a topology control algorithm. With this, one may implement LMST, KNEIGH or any other topology-control algorithm: all would be models of the TC concept.

For a more detailed description and examples on the definition of concepts and models, we address to reader to [2].

In programming terms, an algorithm model implementing a concept is basically a template expecting various parameters. These parameters can be both Operating System (OS) facets and data structures. OS facets represent the connection to the underlying operating system or firmware (e.g., the sensor radio or the timers), thus being an abstraction layer to the OS, and thus also to the hardware platform. The concepts and models of this abstraction layer form the so-called *external interface* (see Fig. 3.2). OS facets are passed to an algorithm as template arguments and the compiler later resolves such calls to the OS. An OS facet can be implemented by different models, that may have different advantages or purposes. The user can pass any of those models to an algorithm at compile time, and no extra overhead is payed for that. Additionally, the so-named *internal interface* is formed by data structure models and concepts, which make the algorithms independent from specific implementations for their required data structures. As part of this internal interface, Wiselib provides the pMP and the pSTL. The pMP is a C-based implementation of big-number operations. The pSTL is an implementation of parts of the STL that does neither use dynamic memory nor exceptions nor RTTI. The pSTL includes nowadays implementations for the data structures `map`, `vector` and `list`, and we recently also included `graph`, `queue` and `priority_queue` and an almost-complete implementation of STL algorithms following the same restrictions as the rest of the pSTL. As for the models of the internal interface, the user can pass any of the models for data structures to an algorithm at compile time.

In general, this separation of concepts and their implementation through a template-based approach leads to a highly flexible and powerful design, because the necessities,

strength and weaknesses of different hardware platforms or different data structures can easily be utilized just by changing the parameters, or even simply by changing makefile targets.

Hardware	Firmware/OS	CPU	Language	Dynamic Memory	ROM Size	RAM Size	Bits
iSense	iSense-FW	Jennic	C++	Physical	128kB	92kB	32
SCW MSB	SCW-FW	MSP430	C	None	48kB	10kB	16
SCW ESB	SCW-FW	MSP430	C	None	60kB	2kB	16
Tmote Sky	Contiki	MSP430	C	Physical	48kB	10kB	16
MicaZ	Contiki	ATMega128L	C	Physical	128kB	4kB	8
TNOde	TinyOS	ATMega128L	nesC	Physical	128kB	4kB	8
iMote2	TinyOS	Intel XScale	nesC	Physical	32MB	32MB	32
GumStix	Emb. Linux	Intel XScale	C	Virtual	16MB	64MB	32
Desktop PC	Shawn	various	C++	Virtual	unlimited	unlimited	32/64
Desktop PC	TOSSIM	(ATMega128L)	nesC	(Physical)	unlimited	unlimited	(8)

Table 3.1: Wiselib target platforms [2]

The abstraction of the OS and hardware platform provided by the external interface is used by Wiselib to provide single implementations of its algorithms that can be run on heterogeneous test beds. In its current version, Wiselib supports nine different sensor types, namely, iSense nodes, SCW MSB, SCW ESB, Timote Sky, MicaZ, TNOde, iMote2, GumStix, and desktop PCs. Each of these sensor nodes have a different micro-controller type, its own operating system, its more prominent programming language, its kind of dynamic memory, its amount of ROM and RAM, and its bit width (see Table 3.1). Observe that two simulator platforms are also considered (Shawn and TOSSIM).

3.2 The topology-control concept

As said, concepts and models are the fundamental design principle in Wiselib. In this document we are concerned with topology control algorithms, so that our main concept is *TopologyControl*. This concept will have multiple models, one per algorithm: LMST, KNEIGH, etc. The *TopologyControl* concept will not only use other concepts from the external and internal interface, but also from other categories of algorithms it works in conjunction with, when necessary (e.g., localization algorithms).

Specifically, the interface of the *TopologyControl* concept is summarized in the following "class":

```
concept TopologyControl {
  typedef ... Neighbors;
  int enable(void);
  int disable(void);
  const Neighbors& topology() const;
  template<class T, void (T::*TMethod)()>
    int reg_listener_callback (T* obj_pnt);
  template<void (*TMethod)()>
    int reg_listener_callback ();
  void unreg_listener_callback (int);
};
```

In order to start a TC algorithm, all nodes call their *enable()* method to run the distributed algorithm in the background. Nodes can register listeners in order to be notified about changes in the topology. This implementation of the Observer pattern [5] is achieved by appropriately calling the template-shaped *reg_listener_callback()* methods, which can register both static functions and member methods as callbacks. In these callbacks, nodes can query the current topology using the *topology()* method, which returns a container that stores the identities of the neighbours of the node. A node can quit a TC algorithm by calling the *disable()* method, and previously registered callbacks can be removed using the method *unreg_listener_callback()*.

Let us remark that the full topology information is not known by the nodes. It is only local, as all the nodes only have knowledge of their own neighbors, not of the whole graph. Also, the task of lowering the transmission range in order to strictly communicate with the reported neighbors has been factored out of this concept. The user is expected to set the transmission power before sending messages. Indeed, there exist many different ways to do so, and these are not tied to the TC algorithm.

As a sample implementation of the previous TC concept, consider the following specific code for the XTC protocol:

```
template<class OsModel_P,
  typename OsModel_P::size_t MAX_NODES=32,
  class Radio_P = typename OsModel_P::Radio,
  class Timer_P = typename OsModel_P::Timer>
class XTCProtocol: public TopologyBase<OsModel_P> {
public: ...
    typedef vector_static <node_id_t, MAX_NODES>
        Neighbors;
    ...
}
```

The *XTCProtocol* is parameterized using the following template parameters: *OSModel_P* is the model that implements the operating system concept. *MAX_NODES* is a constant value that bounds the number of neighbors of a single node. *Radio_P* and *Timer_P* are, respectively, the models implementing radio and timer concepts. For instance, the radio defaults to the default radio model provided by the operating system model, which gives raw access to the radio facilities in the node firmware. An alternative would be to use the *VirtualLinkRadio* model, which support transparent communication across different test beds using Internet links.

To avoid code redundancies, ease maintenance and simplify the implementation, all our TC models also inherit from the *TopologyBase* template-shaped base class. We have used this base class to implement all functionality related to callback management, which is common to all implementations.

As the *MAX_NODES* parameter and the *Neighbors* type reflect, this particular model only uses static memory. The rationale behind this decision is that some platforms where Wiselib is intended to be deployed (see Table 3.1) do not support dynamic memory or, when they do, incur into fragmentation issues. Therefore, we have preferred not to use dynamic memory as long as possible and stick with the pSTL data structures, whose size must be given at compilation time.

TC algorithms that need knowledge about the coordinates of the nodes are parameterized with an extra parameter that describes the localization algorithm to use. For

instance, the declaration of the LMST model is as follows:

```
template<typename OsModel_P,
    typename Localization_P,
    typename OsModel_P::size_t MAX_NODES = 32,
    typename Radio_P=typename OsModel_P::Radio,
    typename Timer_P=typename OsModel_P::Timer>
class LMSTProtocol: public TopologyBase<OsModel_P> {
    ...
}
```

Observe that the LMST model is parameterized with *Localization_P*, which references a concept for a localization algorithm. That algorithm is used by LMST to compute the position of the sensor nodes. Localization algorithms are another category of algorithms included in Wiselib. This category of algorithms is still not implemented, but scheduled for this year. Thus, we just implemented a fake localization algorithm where the coordinates of the nodes are hard-coded with their positions in our test bed.

The TC algorithms FLSS, KNEIGH, XTC and CBTC have been implemented following the same principles as described for LMST. Their particularities, as the reader knows by now, are supported by the models of concepts from the external and internal interface.

3.3 Topology-control models

In this section we discuss the implementation decissions taken when designing the different models of the TC concept.

3.3.1 LMST

```
template<typename OsModel_P,
    typename Localization_P,
    typename OsModel_P::size_t MAX_NODES = 32,
    typename Radio_P=typename OsModel_P::Radio,
    typename Timer_P=typename OsModel_P::Timer>
class LMSTProtocol: public TopologyBase<OsModel_P> {
    ...
  void set_startup_time ( millis_t );
  void set_work_period ( millis_t );
}
```

LMST protocol requires knowledge of the position of the different nodes. As said in the previous section, we added a template parameter that allows to specify a localization algorithm to use for this purpose.

The optional phase of constructing a graph with only bidirectional links has been implemented.

`MAX_NODES` template parameter is used to specify the size of the static data structures.

`set_startup_time` allows specifying a delay before starting the protocol after enabling it, waiting for all nodes to start up.

`set_work_period` allows specifying how often the topology-generation protocols is executed to update the topology with possible changes, such as node mobility or nodes going off-line.

3.3.2 KNeigh

```
template<class OsModel_P,
  typename OsModel_P::size_t K = 9,
  class Distance_P=uint16_t ,
  class Radio_P = typename OsModel_P::Radio,
  class Timer_P = typename OsModel_P::Timer,
  class Rand_P = typename OsModel_P::Rand>
  class KneighProtocol: public TopologyBase<OsModel_P> {
  ...
  void set_delta ( millis_t );
  millis_t delta () const;
  void set_d ( millis_t );
  millis_t d() const;
  void set_tau ( millis_t );
  millis_t tau () const;

  static void set_default_delta ( millis_t );
  static millis_t default_delta ();
  static void set_default_d ( millis_t );
  static millis_t default_d ();
  static void set_default_tau ( millis_t );
  static millis_t default_tau ();
}
```

The Kneigh protocol requires knowledge of the distance between nodes. In this implementation, we estimate the distance based on signal strength measures.

`K` is the upper bound to the node degree this protocol is based on. It is also used to set the size of static data structures.

`Distance_P` allows specifying the data type to represent distances.

A set of methods allows specifying the values of the timing parameters Δ, d and τ of the algorithm, and their default values for new instances of the algorithm.

The optional prune stage of the protocol was chosen not to be implemented, as it requires estimating transmission power based on distance estimation. This is not expected to be accurate enough in a real situation for the prune stage to be reliable.

3.3.3 XTC

```
template<class OsModel_P,
  typename OsModel_P::size_t MAX_NODES=32,
  class Radio_P = typename OsModel_P::Radio,
  class Timer_P = typename OsModel_P::Timer>
  class XTCProtocol: public TopologyBase<OsModel_P> {
  ...
  void set_delta ( millis_t );
  millis_t delta () const;

  static void set_default_delta ( millis_t );
```

```
    static  millis_t  default_delta ();
}
```

In order to generate the neighbour order of the algorithm, the first step is to send a broadcast message that allow nodes to detect the set of immediate neighbours. Each node, then, establishes the order based on signal strength measures.

`MAX_NODES` template parameter is used to specify the size of the static data structures.

A set of methods allows specifying the value of a parameter Δ which is an upper bound to the time that all nodes in the network take to start (equivalente to the Δ parameter of the KNeigh protocol).

3.3.4 CBTC

```
template<class OsModel_P,
  typename Localization_P,
  class Radio_P = typename OsModel_P::Radio,
  typename OsModel_P::size_t MAX_NODES=32>
  class CbtcTopology: public TopologyBase<OsModel_P> {
  ...
  void set_startup_time ( millis_t );
  void set_work_period_1 ( millis_t );
  void set_work_period_2 ( millis_t );
  void set_alpha ( double );
}
```

CBTC protocol requires directional information. This support is not implemented in the wiselib, as no supported platforms have directional antennas. The direction information is, then, calculated based on position information. Position information is gathered using a localization algorithm specified by (Localization_P) template parameter.

`MAX_NODES` template parameter is used to specify the size of the static data structures.

`set_startup_time` allows specifying a delay before starting the protocol after enabling it, waiting for all nodes to start up.

`set_work_period_1` allows specifying the delay used in the first phase of the algorithm to wait between transmissions at different power levels.

`set_work_period_2` allows specifying time period of the second phase of the algorithm.

`set_alpha` allows specifying the angle of the cone to be used by the protocol. Optimizations are automatically set accordingly to this parameter.

3.3.5 FLSS

```
template<typename OsModel_P,
    uint16_t K,
    typename Localization_P,
    uint16_t MAX_NODES,
    typename Radio_P=typename OsModel_P::Radio,
    typename Timer_P=typename OsModel_P::Timer>
  class FlssTopology: public TopologyBase<OsModel_P> {
  ...
```

```
    void set_startup_time ( millis_t );
    void set_work_period ( millis_t );
}
```

FLSS protocol generates a k-connected topology. This configurable k is specified via the template parameter `K`.

Edge weight of the graph is implemented to be node distance. The distance is calculated from node position information, estimated by the template-specified localization algorithm.

`MAX_NODES` template parameter is used to specify the size of the static data structures.

`set_startup_time` allows specifying a delay before starting the protocol after enabling it, waiting for all nodes to start up.

`set_work_period` allows specifying how often the topology-generation protocols is executed to update the topology with possible changes, such as node mobility or nodes going off-line.

3.4 Topology-constrained routing

As a proof of concept and for testing purposes, a routing protocol has been implemented that takes as input the topology generated by a topology-control protocol and restricts routing to the given topology.

3.4.1 Controlled-topology tree routing

```
template<typename OsModel_P,
         typename Timer_P = typename OsModel_P::Timer,
         typename Radio_P = typename OsModel_P::Radio,
         typename Debug_P = typename OsModel_P::Debug,
         typename OsModel_P::size_t MAX_NEIGHBORS=32>
class ControlledTopologyTreeRouting
    : public RoutingBase<OsModel_P, Radio_P> {
...
    typedef vector_static <OsModel,node_id_t,MAX_NEIGHBORS> Neighbors;
...
    void enable ();
    void disable ();
    void set_sink ( bool );
    void send( node_id_t , size_t , block_data_t *);
    void set_topology (Neighbors *)
    Neighbors *topology ();
  ...
}
```

This is a tree-routing protocol. It routes messages from any node in the network to a given root node called sink.

`set_sink` is used to specify which concrete node is the sink.

The sink broadcasts a message periodically including the hop count to the sink (obviously zero). Nodes that receive this message set the sink as their parent in the tree and start sending the broadcast message themselves (with updated hop count). When a node that is already connected to a parent receives a new broadcast messages with a

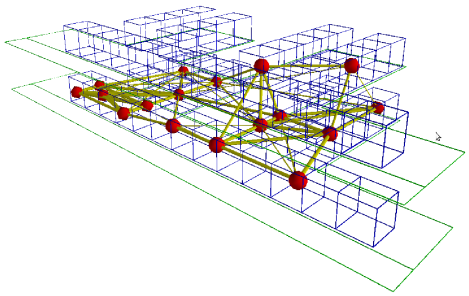

Figure 3.3: Representation of the UPC test bed. The width of the links is proportional to their measured capacity at full power. Failing links are not shown.

lower hop count as the parent, it updates its parent to the new one. This process goes on while the protocol is enabled, adapting to changes in the network.

`send` is used in a connected node to send messages to the sink.

This is the standard tree routing behaviour that this protocol follows when no topology has been specified (or a null pointer has been specified).

`set_topology` can be used to specify a concrete topology to use for routing, probably the result of a topology-control protocol. The only difference in behaviour when a topology has been specify is that, when receiving a broadcast message, if the source of the message is not in the topology, the message is just ignored. It is, hence, not possible to establish a link that does not belong to the topology.

3.5 Topology control in UPC testbed

Wisebed infrastructure consists of several federated test beds of various sizes (dozens to hundreds of nodes), currently including nine test beds that amount to 600 nodes. This infrastructure is inherently heterogeneous, using hardware with different computational resources and sensing capacities (see Table 3.1). The Wisebed federated test beds are interconnected together using the TARWIS system [6], which provides a web-based GUI for test bed management, experiment configuration, and experiment monitoring.

Currently, the UPC test bed consists of 17 iSense nodes from Coalesenses GmbH. The iSense nodes use a Jennic JN5139, a solution that combines the controller and the wireless communication transceiver in one chip. The controller has a 32-bit RISC architecture and runs at 16MHz. It offers 96kB of RAM and 128kB of Serial Flash. It has a transmit power from -30dBm to +0dBm, which reaches ranges of up to 500m in free air.

The 17 nodes are located in our departmental building. All our nodes are currently in the same level, except two nodes that are on another level. In this test bed, sensor nodes in the entire network are not reachable by each other. Because of the contrived architecture of our building, their range is strongly shortened and some sensors must communicate with others using multiple hops. Fig. 3.3 depicts the structure of our building, the location of the nodes and their topology at full power.

We have tested in UPC test bed TC algorithms developed for the Wiselib. To do so,

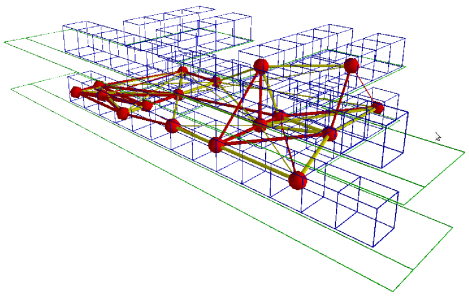

Figure 3.4: Topology computed by the LMST algorithm (emphasized with red links).

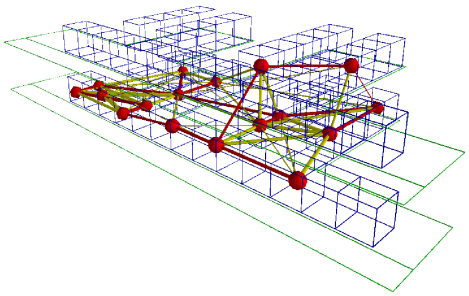

Figure 3.5: Topology computed by the XTC algorithm (emphasized with red links).

we have written a testing application that started the computation of each TC algorithm and used a tree routing algorithm to gather the neighbors of each node to a server, where the result was stored and post-processed.

For instance, Fig. 3.4 and Fig. 3.5 show the resulting topologies computed by the LMST and XTC algorithms, respectively. In both figures, red links mark the links selected by the topology algorithm.

Chapter 4

Comparison of topology-control protocols

In this chapter we perform both theoretical and experimental comparison of a selection of the topology-control protocols implemented in the Wiselib: LMST, KNeigh, XTC and CBTC. These were the first ones to be implemented, are the most stable ones and also the most important and representative.

4.1 Theoretical comparison

In this section we discuss the advantages and drawbacks of each algorithm.

4.1.1 LMST

Advantages

- Connectivity is preserved.
- Node degree is bounded by 6.
- The resulting topology has few cycles, it is almost a tree.

Disadvantages

- Position information of each node is needed.
- It does not take into account the obstacles and characteristics of the environment that can limit communications.
- Two relatively close nodes can have very long communication paths.
- It needs a special phase to select only bidirectional links.

4.1.2 KNeigh

Advantages

- Node degree is bounded by selectable constant K.

Disadvantages

- It requires that nodes are stationary.
- It requires same maximum transmission power for all nodes.
- An estimation of distance between nodes is needed. This is a weaker requirement than having position information, as positions allow to calculate distances.
- An unreliable estimation of distances requires a greater value of K.
- Difference between node wakeup times must be bounded by a known constant.
- It needs a special phase to select only bidirectional links.
- Connectivity preservation is not warranted, just with high probability.

4.1.3 XTC

Advantages

- Connectivity is preserved.
- Generated links are bidirectional.
- It does not require position, distance nor direction information. Any measure of link quality is good enough.
- Depending on the link quality measure, it can handle obstacles and characteristics of the environment that limit communications.
- If nodes are located in an Euclidean plane, node degree is bounded by 6 and the resulting graph is planar.
- Cycles are of minimum length 4.

Disadvantages

- If nodes are not located in an Euclidean plane, node degree is not bounded.

4.1.4 CBTC

Advantages

- If $\alpha \leq 5\pi/6$, connectivity is preserved.
- It tests for needed transmission power to arrive to each node. This is more precise and realistic than other approaches.

Disadvantages

- If $\alpha > 5\pi/6$, connectivity is not necessarily preserved.
- Requires direction information. This is a weaker requirement than position information, as direction can be calculated from positions.
- It requires a special phase to generate only bidirectional links.
- If unidirectional links are removed, $\alpha \leq 2\pi/3$ is required to preserve connectivity.
- It requires sending more messages than other protocols.

4.2 Experimental comparison

An experiment with real devices in real conditions has been designed and performed in order to have experimental data and comparison of the protocols. To the best of our knowledge, until now, they had only been tested and compared in simulations.

4.2.1 Experiment design

The idea of the experiment is to execute topology-control algorithms in real nodes, adjust transmission power to the minimum needed to arrive to the selected neighbours and then measure battery consumption while sending messages at the adjusted transmission power.

Two variations have been tested. The first proceeds as follows:

- Execute the topology-control protocol till the first topology is generated.
- Test for needed power to arrive to all neighbours.
- Adjust transmission power to the value of the previous step.
- Continuously broadcast a selectable number of maximum-length messages.
- Measure battery consumption.

This variation is a laboratory scenario, not realistic, designed to be as sensible as possible to differences in transmission power.

The second variation proceeds as follows:

- Execute the topology-control protocol till the first topology is generated.
- Test for needed power to arrive to all neighbours.
- Adjust transmission power to the value of the previous step.
- Enable ControlledTopologyTreeRouting specifying the generated topology. The sink will change in every execution of the experiment.
- Send a selectable number of minimum-length messages to the sink, with a small random delay between one message and the next.
- Measure battery consumption.

This variation is designed to be more similar to real conditions in WSN's. Messages must be short and random delays must be inserted to minimize collisions. As a consequence, this variation is much less sensible to differences in transmission power, depending much on the characteristics of the hardware platform.

In both cases, the experiment is executed, for each condition, eight times. In the second variation, a different sink node will be used for tree routing in every execution.

4.2.2 Phases of the experiment

The experiment is designed to be executed in two phases.

The first phase does not use a real topology-control protocol. Instead, a basic topology algorithm detects all accessible nodes in the immediate neighbourhood at a given preset transmission power. The transmission-power adjustment step is omitted.

The control condition is performing the test at maximum power.

The experimental condition is performing the test at minimum power.

If (and only if) there are statistically-significant differences in battery consumption in this first phase, the second phase is performed.

In the second phase, there are two control conditions (the same conditions as in the first phase, that will act as lower and upper bounds to battery consumption), and four experimental conditions, one for each tested protocol.

This second phase allows to compare protocols between each other and with the control conditions.

4.2.3 Hardware

We used 8 iSense sensor nodes, identical to the ones in UPC testbed, described in a previous chapter, with energy modules and identical $3.6V$, $2250mAh$ batteries.

Sensors were deployed in different rooms, with realistic conditions, on a total surface of about $60m^2$.

4.2.4 Software

Basic-topology protocol

```
template<class OsModel_P,
  class Neigh_P,
  class Radio_P = typename OsModel_P::Radio,
  class Timer_P = typename OsModel_P::Timer>
  class BasicTopology: public TopologyBase<OsModel_P> {
  ...
  void set_ping_period ( millis_t );
   millis_t ping_period () const;
  static void set_default_ping_period ( millis_t );
  static millis_t default_ping_period ();
}
```

This protocol implements the topology-control concept. A ping message is periodically broadcasted. Received messages identify neighbour nodes.

Template parameter `Neigh_P` specifies the data structure that stores the resulting topology.

A set of methods allow specifying ping period and its default value.

Adjusting transmission power

```
template<class OsModel_P,
  class Radio_P = typename OsModel_P::TxRadio,
  class Neigh_P = vector_static <OsModel_P, typename Radio_P::node_id_t,MAX_NEIGH_DEF>,
  class Timer_P = typename OsModel_P::Timer>
class SequentialTry {
  ...
  void enable ();
  void disable ();

  TxPower power() const;

  void set_neighbors (Neighbors &);
  Neighbors &neighbors();

  void set_delta ( millis_t );
  millis_t delta ();

  template<class T, void(T::*TMethod)()>
  void reg_listener_callback (T *);

  template<void(*TMethod)()>
  void reg_listener_callback ();

  void unreg_listener_callback ();

  static void set_default_delta ( millis_t );
  static millis_t default_delta ();

}
```

There is no defined concept, yet, for this kind of algorithm. `SequentialTry` sequentially sets different allowed transmission-power values, from minimum to maximum, and sends ping messages. It also periodically sends pong messages with a list of node id's from which it has received ping messages. These pong messages are always sent at maximum power. The protocol proceeds until it gets pong messages from all neighbour nodes in the given topology. The last transmission power tested is the minimum power to reach all neighbour nodes in the topology and is the result of the algorithm.

`power` provides the power value found by the protocol.

`set_neighbors` is used to set the topology to test.

A set of methods allow specifying the Δ value, which is the upper bound of the differences between the starting times of the different nodes in the network. It is equivalent to the same parameter in the KNeigh protocol.

Main applications

There are two main application for this experiment. One executes the experiment itself and sends the results to a given node. The second is the application that receives the results from the different nodes in the network and logs them to the computer. An additional sensor node was used for this purpose, attached to the computer via a gateway module and USB port.

Both applications are native iSense applications using the platform-independent facilities provided by the Wiselib.

Measuring battery consumption

The main application of the experiment need to measure battery-charge consumption. To methods have been developed for this purpose, corresponding to two measures that iSense firmware makes available.

The first uses a firmware function that gets the current battery charge estimated by the energy module, in an integer number of μAh. The resolution of this measure is $30\mu Ah$. The application stores the battery capacity at the beginning of sending messages and subtracts this value from the battery capacity at the end of the process.

The second method uses a firmware function that gets the mean battery current over periods of $3.5s$, in μA. The application initializes a counter at zero before starting sending messages and then every exactly $3.5s$ it reads the battery current and adds it to the counter. When the process finishes, the total value in the counter is multiplied by 3.5 resulting in the total battery charge consumed in μAs.

Both methods of measuring consumption are expected to give similar results.

4.2.5 Results

In this subsection we give the statistical analysis of the resulting data from the execution of the experiment.

Resulting values for each node were added to give the values of the dependent variables.

We have analysed two dependent variables, named Capacity and Current. Capacity is the consumption measured by the first method, based in measuring battery capacity at the beginning and the end of the experiment. Its units are μAh. Current has been generated based on the values obtained from mean battery current measures. It is the result of dividing the obtained values in μAs by 3600 to give μAh. Three decimals have been used to preserve the same number of significant digits.

The independent variable, named Algorithm, is a numeric categorical variable whose values mean:

- 1: Basic topology at maximum power.
- 2: Basic topology at minimum power.
- 3: LMST.
- 4: KNeigh.
- 5: XTC.
- 6: CBTC.

The data was analysed using One-way analysis of variance. Test were performed to check that the data accomplishes the assumptions of the model. The assumptions are:

- Cases are independent.
- Measures are normally distributed.
- Variances of each group are equal.

First variation

Here we give the results for the first variation of the experiment, the unrealistic one that constantly broadcasts messages. We first test the assumptions of the ANOVA model. The results are given in figure 4.1.

The significances of all test of normality, for both variables and both conditions lead to accept that measures follow a normal distribution.

The significances of the test of homogeneity of variances, for both variables, lead to accept that variances are equal.

Last to test for independence of observations. We apply the runs test in [9] using the median of the observations. The results for both variables are as follows:

$$- + + - - - + + - - + - - + + +$$

$$n_1 = n_2 = 8$$

$$r = 8$$

In table A.12 in [10] we can check that for those values of n_1 and n_2 and an error level of 0.05, r values can be between 4 and 13. Hence, we accept that the observations are independent.

We now can proceed with the analysis of variance. The results are in figure 4.2.

There were clearly no significant differences between both experimental conditions, for both variables. There is no reason to proceed with phase 2 of the experiment.

Statistics

		Capacity	Current
N	Valid	16	16
	Missing	0	0
Median		150000,00	150363,488

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Algorithm	Statistic	df	Sig.	Statistic	df	Sig.
Capacity	1	,237	8	,200*	,843	8	,081
	2	,112	8	,200*	,997	8	1,000
Current	1	,264	8	,106	,854	8	,104
	2	,116	8	,200*	,987	8	,989

a. Lilliefors Significance Correction
*. This is a lower bound of the true significance.

Test of Homogeneity of Variances

	Levene Statistic	df1	df2	Sig.
Capacity	,009	1	14	,927
Current	,083	1	14	,778

Page 1

Figure 4.1: Tests of the assumptions of the ANOVA model for the first variation of the experiment, phase 1

Descriptives

		N	Mean	Std. Deviation	Std. Error
Capacity	1	8	149765,63	1207,191	426,806
	2	8	150078,13	1477,416	522,345
	Total	16	149921,88	1313,293	328,323
Current	1	8	149976,165	1125,53141	397,935447
	2	8	150390,004	1466,96142	518,649186
	Total	16	150183,085	1281,05763	320,264409

Descriptives

		95% Confidence Interval for Mean		Minimum	Maximum
		Lower Bound	Upper Bound		
Capacity	1	148756,39	150774,86	148500	151250
	2	148842,97	151313,28	147750	152500
	Total	149222,07	150621,68	147750	152500
Current	1	149035,197	150917,133	148745,725	151452,388
	2	149163,594	151616,415	147773,013	152541,419
	Total	149500,457	150865,712	147773,013	152541,419

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Capacity	Between Groups	390625,000	1	390625,000	,215	,650
	Within Groups	25480468,7	14	1820033,48		
	Total	25871093,7	15			
Current	Between Groups	685052,458	1	685052,458	,401	,537
	Within Groups	23931577,4	14	1709398,39		
	Total	24616629,9	15			

Figure 4.2: Results of ANOVA for the first variation of the experiment, phase 1

Second variation

Here we give the results for the second variation of the experiment, the realistic one that sends messages through tree routing. We first test the assumptions of the ANOVA model. The results are given in figure 4.3.

The significances of all test of normality, for both variables and both conditions lead to accept that measures follow a normal distribution.

The significances of the test of homogeneity of variances, for both variables, lead to accept that variances are equal.

Last to test for independence of observations. We apply the runs test in [9] using the median of the observations. The results for both variables are as follows:

$$+\ +\ -\ -\ -\ +\ +\ -\ +\ +\ -\ -\ +\ +\ -\ -$$

$$n_1 = n_2 = 8$$

$$r = 8$$

In table A.12 in [10] we can check that for those values of n_1 and n_2 and an error level of 0.05, r values can be between 4 and 13. Hence, we accept that the observations are independent.

We now can proceed with the analysis of variance. The results are in figure 4.4.

There were clearly no significant differences between both experimental conditions, for both variables. There is no reason to proceed with phase 2 of the experiment.

Statistics

		Capacity	Current
N	Valid	16	16
	Missing	0	0
Median		212125,00	212491,992

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Algorithm	Statistic	df	Sig.	Statistic	df	Sig.
Capacity	1	,187	8	,200*	,926	8	,477
	2	,206	8	,200*	,919	8	,420
Current	1	,157	8	,200*	,950	8	,716
	2	,179	8	,200*	,921	8	,439

a. Lilliefors Significance Correction
*. This is a lower bound of the true significance.

Test of Homogeneity of Variances

	Levene Statistic	df1	df2	Sig.
Capacity	,013	1	14	,911
Current	,000	1	14	,994

Figure 4.3: Tests of the assumptions of the ANOVA model for the second variation of the experiment, phase 1

Descriptives

		N	Mean	Std. Deviation	Std. Error
Capacity	1	8	211734,38	3114,939	1101,297
	2	8	212314,00	3071,764	1086,032
	Total	16	212024,19	3003,486	750,871
Current	1	8	211999,445	3136,85392	1109,04534
	2	8	212668,198	3088,44245	1091,92930
	Total	16	212333,821	3026,95919	756,739797

Descriptives

		95% Confidence Interval for Mean		Minimum	Maximum
		Lower Bound	Upper Bound		
Capacity	1	209130,22	214338,53	207375	215500
	2	209745,94	214882,06	208750	217012
	Total	210423,74	213624,63	207375	217012
Current	1	209376,969	214621,920	207438,251	216032,965
	2	210086,196	215250,201	208969,028	217146,861
	Total	210720,869	213946,774	207438,251	217146,861

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Capacity	Between Groups	1343860,56	1	1343860,56	,140	,713
	Within Groups	1,340E8	14	9569289,13		
	Total	1,353E8	15			
Current	Between Groups	1788923,34	1	1788923,34	,185	,674
	Within Groups	1,356E8	14	9689164,69		
	Total	1,374E8	15			

Figure 4.4: Results of ANOVA for the second variation of the experiment, phase 1

Chapter 5

Conclusions and future work

The results of the experiments are somehow surprising, but indeed confirm the need for experimentation in real platforms.

We have to conclude that, in the described hardware platform, there are no significant differences in energy consumption due to message transmission power. Probably, energy consumption by the power amplifier is not significant when compared with overall consumption of the rest of the components of the radio.

This looks to be related to radio frequency. As a rule of thumb, the higher the frequency, the less important is consumption of the power amplifier. We can predict that this might happen in most devices using a standard $2.4GHz$ Wi-Fi radio.

Until now, topology-control protocols were consider crucial to reduce energy consumption and extend network lifetime. Now we have to consider that its significance is relative and depends on concrete hardware platforms. Of course, topology control can still be useful in all hardware platforms to reduce radio interference and increase network capacity and reliability.

More exhaustive research should be performed in future using different hardware platforms to first distinguish between the three following cases:

- Differences in transmission power are not significant when compared to overall device consumption.
- Differences in transmission power are significant in laboratory conditions when constantly transmitting data, but are not significant in more realistic scenarios when most of the time devices are idle or receiving messages. This could happen, for example, if consumption is not significantly higher when transmitting than when idle or receiving.
- Differences in transmission power are significant in realistic scenarios.

Once determined what devices are in the third case, these could be used to perform a relative comparison between the different topology-control protocols.

Other objectives for the future are to stabilize and finish implementing all available topology-control protocols in the Wiselib and to perform research on other ways to save energy, for example, by reducing the duty cycle, and implementing in the Wiselib the already-available alternatives to topology-control.

Bibliography

[1] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addision-Wesley, 2001.

[2] T. Baumgartner, I. Chatzigiannakis, S. Fekete, C. Koninis, A. Kröller, and A. Pyrgelis. Wiselib: A generic algorithm library for heterogeneous sensor networks. In *7th European Conference on Wireless Sensor Networks*, 2010.

[3] D. Blough, M. Leoncini, G. Resta, and P. Santi. The k-neighbors protocol for symmetric topology control in ad hoc networks. In *4th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing*, pages 141–152, 2003.

[4] I. Chatzigiannakis, S. Fischer, C. Koninis, G. Mylonas, and D. Pfisterer. WISEBED: an open large-scale wireless sensor network testbed. In *1st Intl. Conference on Sensor Networks Applications, Experimentation and Logistics*, Lecture Notes of the Institute for Computer Sciences, Social-Inf, 2009.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts, 2000.

[6] P. Hurni, T. Staub, G. Wagenknecht, M. Anwander, and T. Braun. A secure remote authentication, operation and management infrastructure for distributed wireless sensor network testbeds. *Electronic Communications of the EASST*, 17, 2009.

[7] L. Li, Y. W. J. Halpern, P. Bahl, and R. Wattenhofer. Analysis of a cone-based distirbuted topology control algorithm for wireless multi-hop networks. In *20th ACM Symposium on Principles of Distributed Computing*, pages 264–273, 2001.

[8] N. Li and J. Hou. A fault-tolerant topology control algorithm for wireless sensor networks. In *10th Intl. ACM Conference on Mobile Computing and Networking*, pages 275–286, 2004.

[9] P. Lubin, M. A. Maciá, and P. Rubio. *Psicología Matemática II*, volume 2, chapter 12, pages 41–42. UNED, second edition, Aug. 2000.

[10] P. Lubin, M. A. Maciá, and P. Rubio. *Psicología Matemática II*, volume 3, pages 358–362. UNED, second edition, Aug. 2000.

[11] N. Li, J. C. Hou, and L. Sha. Design and analysis of an MST-based topology control algorithm. *Proc. IEEE INFOCOM 2003*, 3:1702–1712, 2003.

[12] R. W. L. L. P. Bahl and Y. Wang. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In *20th Joint Conference of the IEEE Computer and Communications Societies*, pages 1388–1397, 2001.

[13] V. Rodoplu and T. Meng. Minimum energy mobile wireless networks. *IEEE Journal Selected Areas in Communication*, 17(8):1333–1344, 1999.

[14] A. K. S. F. S. Fekete and D. Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *4th Intl. Conference on Networked Sensing Systems (INSS)*, page 299, 2007.

[15] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, 2005.

[16] P. Santi. *Topology control in wireless ad hoc and sensor networks*. Wiley, 2005.

[17] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2000.

[18] D. Vandevoorde and N. Josuttis. *C++ Templates: The Complete Guide*. Addision-Wesley, 2003.

[19] R. Wattenhofer and A. Zollinger. XTC: A practical topology control algorithm for ad hoc networks. In *18th Intl. Parallel and Distributed Processing Symposium*, 2004.

[20] WISELIB. http://www.wiselib.org.