Self-Adapting Maxflow Routing Algorithm for WSNs: Practical Issues and Simulation-Based Assessment

Andrea Seraghiti
University of Urbino
61029 Urbino, Italy
andrea.seraghiti@uniurb.it

Saverio Delpriori
University of Urbino
61029 Urbino, Italy
delprior@sti.uniurb.it

Emanuele Lattanzi
University of Urbino
61029 Urbino, Italy
lattanzi@sti.uniurb.it

Alessandro Bogliolo
University of Urbino
61029 Urbino, Italy
alessandro.bogliolo@uniurb.it

ABSTRACT

Autonomous wireless sensor networks are subject to power, bandwidth, and resource limitations that can be represented as capacity constraints imposed to their equivalent flow networks. The maximum sustainable workload of a sensor network (i.e., the maximum data flow from the sensor nodes to the collection point which is compatible with the capacity constraints) is the *maxflow* of the flow network.

This paper presents a self-adapting maxflow routing algorithm which is able to route any sustainable workload while automatically adapting to time-varying operating conditions. The algorithm has been implemented on top of OMNeT++ [1] in order to address practical issues and to enable simulation-based assessment and design exploration. Simulation results demonstrate the effectiveness and the applicability of the proposed approach.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Network Protocols—*Routing protocols*

Keywords

Routing, Self-adaptation, Maxflow, Simulation

1. INTRODUCTION

Sensor nodes are usually designed in order to meet tight constraints in terms of size, cost, and lifetime, which translate into resource constraints imposed to the algorithms running on top of them [15]. Moreover, the wide range of possible applications and operating environments makes scalability and adaptation essential features of *wireless sensor networks* (WSNs) [6].

Since the main task of any sensor network is to gather data from the environment, the routing algorithm applied to the network is one of the most critical design choices, which has a sizeable impact on power consumption, performance, dependability, scalability, and adaptation. In general there are two phases in the operation of a WSN: *dissemination*, which is the diffusion of control information to be used to dynamically change the sampling task (namely, the sampling area, the sampled nodes, the sampling rate, ...) and *collection*, which is the transmission of sampled data from sensor nodes to one or more collection points [10]. Both of them are deeply affected by the networking solutions adopted.

The research efforts devoted to the development of specific routing algorithms for WSNs are documented in many comprehensive surveys [2, 15, 5]. Levis *et al.* point out the emergence of "networking primitives" in WSNs and discuss the efficiency of a "polite gossip" policy to drive both dissemination and collection with a limited control traffic overhead. Energy efficiency is a primary concern in the design of routing algorithms [14, 11]: If the routing algorithm requires too many control packets, chooses sub-optimal routes, or requires too many computation at the nodes, it might end up reducing the lifetime of the network because of the limited energy budget of battery-operated sensor nodes. Lifetime issues can be addressed by means of energy harvesting techniques, which enable the design of autonomous sensor nodes taking their power supply from renewable environmental sources such as sun light and wind [12, 3]. Environmentally-powered systems, however, give rise to additional design challenges due to supply power uncertainty and variability. Moreover, they impose a paradigm shift from *energy-constrained lifetime maximization* to *power-constrained workload optimization*. In fact, as long as the average workload at each node can be sustained by the average power it takes from the environment, the node can keep working for an unlimited amount of time. In this case rechargeable batteries are used only as energy buffers to compensate for environmental power variations, but their capacity does not affect any longer the lifetime of the network.

It has been shown that autonomous wireless sensor networks can be modeled as flow networks [4], and that the *maximum energetically sustainable workload* (MESW) can be deter-

mined by solving an instance of *maxflow* [7, 9]. The solution of maxflow induces a MESW-optimal randomized routing algorithm that can be actually implemented to maximally exploit the available power. Environmental changes, however, impose to periodically recompute the global optimum and to update the routing tables. A distributed version of maxflow has been recently proposed that can be directly executed on the sensor nodes in order to grant to the network the capability of recomputing its own routing tables for adapting to environmental changes [8]. Adaptation, however, is a complex task which conflicts with the normal operations of the sensor network, imposing a trade off between adaptation frequency and availability. In general, the adaptation and scalability needs which are typical of WSNs prompt for the application of some sort of self-organization mechanism [6].

This paper presents a self-adapting maxflow routing algorithm for autonomous wireless sensor networks, which is able to route the maximum sustainable workload under power, bandwidth, and resource constraints, while automatically adapting to time-varying operating conditions. In particular, we start from the theoretical results recently achieved under ideal assumptions [13] and we develop a simulation model (built on top of OMNeT++ [1]) to be used to address implementation issues, to demonstrate the actual applicability of the approach, and to perform design exploration and assessment.

The rest of the paper is organized as follows: Section 2 introduces the network model, which provides a general framework for representing energy, bandwidth and resource constraints; Section 3 presents the self-adapting maxflow routing algorithm and points out its theoretical soundness and its practical issues; Section 4 outlines a simulation model which provides an actual implementation of the algorithm; Section 5 presents preliminary results and draws conclusions.

2. NETWORK MODEL

For our purposes, autonomous WSNs can be modeled as weighted directed graphs with vertices associated with network nodes and edges associated with direct links among them. Vertices v_i and v_j are connected by an edge $e_{i,j}$ iff there is a wireless connection from node i to node j. Each node (say, v) is annotated with two variables: $P(v)$, which represents the environmental power available at that node, and $CPU(v)$, which represents its computational power expressed as the number of packets that can be processed in a time unit. Similarly, each edge (say, e), is annotated with variable $C(e)$, which represents the capacity (or bandwidth) of the link, and variable $E(v, e)$, which represents the energy required at node v to process (or generate) a data packet and to forward it through its outgoing edge e.

The maximum number of packets that can be sent across e in a time unit (denoted by $cap(e)$) is limited by its bandwidth ($C(e)$), by the processing speed of the source node ($CPU(v)$), and by the ratio between the environmental power available and the energy spent to process and send a packet ($P(v)/E(v, e)$). In symbols:

$$cap(e) = \min\{C(e), CPU(v), \frac{P(v)}{E(v, e)}\} \quad (1)$$

The overall number of packets that can be processed by node v in a time unit (denoted by $cap(e)$) is limited by its power budget and by its computational power. In symbols:

$$cap(v) = \min\{CPU(v), \frac{P(v)}{E(v)}\} \quad (2)$$

where $E(v)$ represents the energy per packet under the simplifying assumption that it is independent of the outgoing edge chosen to route the packet.

Equations 1 and 2 show that power, bandwidth, and CPU constraints can be expressed as capacity constraints to be imposed both to the edges and to the vertices of the graph, that becomes a node- and edge-constrained *flow network*. The maximum workload that can be sustained by the network can be determined by solving an instance of maxflow for the corresponding flow network [7].

When node constraints do not depend on the outgoing edge (i.e., when the transmission energy is not dynamically adapted to the length of the link) the node-constrained flow network can be transformed into an equivalent edge-constrained flow network, the maxflow of which can be exactly determined in polynomial time [7]. If the simplifying assumption doesn't hold, node constraints need to be explicitly expressed by Equations 3, where $F(e)$ represents the data flow over edge e, and classical maxflow algorithms [7] cannot be applied.

$$\sum_{e\_exiting\_from\_v} F(e) \leq CPU(v)$$

$$\sum_{e\_exiting\_from\_v} F(e)E(v, e) \leq P(v) \quad (3)$$

Sensor networks are made of 4 types of nodes: *sensors*, which are equipped with transducers that make them able to sense the environment and to generate data packets to be sent to a collection point, *sinks*, which generate control packets and collect data packets, *routers*, which relay packets according to a given routing algorithm, and *sensor-routers*, which exhibit the behavior of both sensors and routers. Without loss of generality, for the sake of simplicity in the following we consider a sensor network with only one sink. Generalization to multi-sink networks can be simply obtained by adding a dummy sink connected at no cost with all the actual sinks.

Figure 1.a shows a hierarchical sensor network of 64 sensors (thin circles), 16 routers (thick circles), and 1 sink (square). Sensors and routers are uniformly distributed over a square region, with the sink in the middle. The communication range of each node is equal to the minimum diagonal distance between the routers (edges are not represented for the sake of simplicity). Shading is used to highlight the sensors that need to be sampled according to a given monitoring task. Figures 1.a and 1.b represent two different monitoring tasks of the same sensor network, hereafter called *corner monitoring* and *edge monitoring*, respectively.

Given a WSN, the environmental conditions, and a monitoring task, the *maximum sustainable workload* (MSW) for the network is the maximum rate at which all the target sensors can be concurrently sampled by the sink without violating power, bandwidth, and CPU constraints. In the following

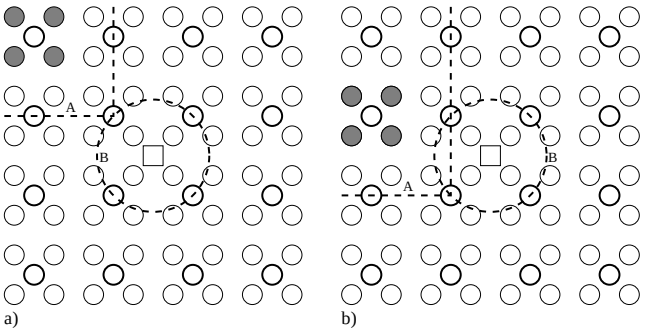

Figure 1: Hierarchical network used as a case study.

we assume the theoretical value of MSW to be determined either by solving an instance of maxflow or by means of iterative simulations [4].

We call *path capacity* of a path $\mathcal{P}$ from node s to node d, denoted by $cap(\mathcal{P})$, the maximum number of packets per time unit that can be routed across the path without violating any node or edge constraint. The *nominal* path capacity is computed a priori by assuming that all the resources along the path are entirely assigned to the path. In practice, it corresponds to the minimum of the capacities of the nodes and edges belonging to the path. The *residual* path capacity, on the contrary, is computed at a given time by taking into account the resources used up to that point.

3. ROUTING STRATEGY

The routing algorithm proposed in this paper implements a simple *greedy* strategy that can be described as follows: *always route packets across the path with maximum residual capacity to the sink.*

According to this strategy, the residual path capacity is used as a routing metric. More precisely, the metric used at node v to evaluate its outgoing edge e is the maximum of the residual capacities of all the paths leading from v to the sink through edge e. The minimum number of hops is used as a second criterion to choose among edges with the same residual path capacity.

As long as all the residual path capacities are kept up to date, the routing strategy grants to the network the capability of routing any sustainable workload while automatically adapting to environmental changes [13]. Since residual path capacities are possibly affected by any routed packet and by any change in the constraints imposed to the nodes and to the edges encountered along the path, in principle routing metrics should be recomputed at each node (and possibly back-propagated) whenever a data packet is processed or an environmental change is detected.

In order to reduce the control overhead of real-time path-capacity re-computation, routing metrics can be kept unchanged for a given time period (hereafter called *epoch*) regardless of traffic conditions and environmental changes, and recomputed only at the beginning of a new epoch. In this way, all the packets processed by a node (say v) in a given epoch are routed along the same path, which is the one with

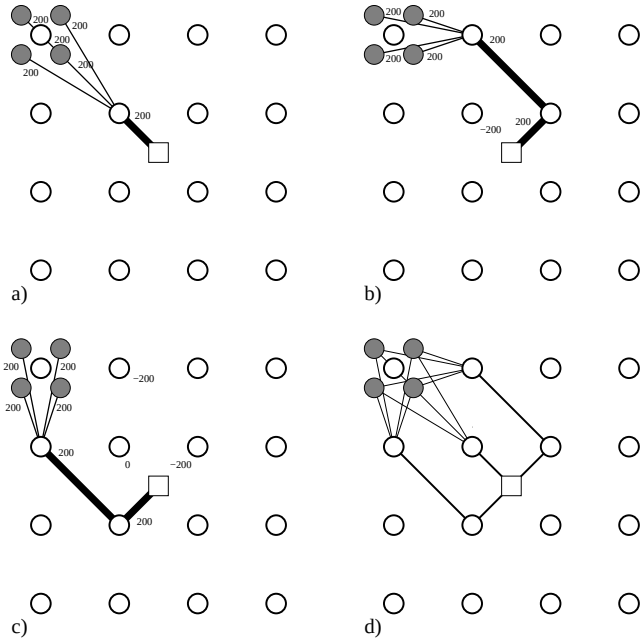

Figure 2: d) Maxflow of the example network, obtained by averaging the flows over three epochs: a), b), and c).

the highest residual capacity at the beginning of that epoch. If the number of packets routed across a link/node in the current epoch exceeds its edge/node capacity, the packets in excess are buffered until next epoch. When recomputing routing metrics at the beginning of the new epoch, buffered packets are treated as capacity debts to be subtracted from nominal capacities in order to compensate for previous over-allocations. It has been shown that the result of this process converges to a cycle-stable routing strategy for any sustainable workload [13].

The periodic behavior of the algorithm is exemplified in Figure 2, which refers to the network of Figure 1.a with the same constraints (namely, $cap(v) = 200$) imposed to sensors and routers, without any edge constraint. For the sake of simplicity, sensor nodes and edges not involved in the monitoring task are not represented in Figure 2. Intuitively, the maxflow is 600, corresponding to a data rate of 150 packets per sensor. In fact, all data packets need to be routed across cut A (shown in Figure 1.a), which contains only 3 routers with an overall capacity of 200x3. An optimal flow distribution is shown in Figure 2.d, where the thickness of each edge represents the flow it sustains: 50 packets per time unit for the thin lines, 200 packets per time unit for the thick ones. Figures 2.a, 2.b, and 2.c show the flows allocated by the self-adapting routing algorithm in three subsequent epochs when the 4 sampled sensors produce 150 data packets per epoch. Residual node capacities at the beginning of each epoch are annotated in the graphs in order to point out the effects of over-allocation. For instance, in the first epoch all data packets (namely, 600) are pushed along the shortest path, which can process only 200. The 400 packets in excess are subtracted from the residual capacity at the beginning of epoch 2, leading to a negative value which imposes to the

```
Node::handleInterest(Interest *msg) {
 1  if (epoch != msg->getEpoch_id())
 2    newEpoch();
 3  new_msg = routeTable->update(msg);
 4  if (new_msg != NULL)
 5    for each gate g != msg->in_gate
 6      send(new_msg,g);
}
```

Figure 3: Pseudocode of `Node::handleInterest()` method.

algorithm the choice of a different path. Since the capacity debt of the shortest path is completely compensated at the end of the third epoch, the entire routing strategy can be periodically applied every three epochs.

The optimal maxflow solution of Figure 2.d can be obtained by averaging the flows allocated by the self-adapting algorithm in the three epochs shown in the figure.

4. SIMULATION MODEL

Although the self-adapting capabilities of the ideal greedy strategy have been theoretically proved independently of its implementation [13], the actual applicability of the epoch-based approach depends on additional features of practical interest, such as control traffic overhead, routing efficiency (number of hops), convergence speed, maximum buffer size, and scalability, that still need to be assessed by means of extensive experiments. To this purpose, the routing algorithm has been implemented on top of OMNeT++, a discrete-event, open-source, modular network simulator [1]. Network components are written in C++ and composed using a high-level network description language called *NED*.

The object-oriented implementation of the routing algorithm is based on two main classes: `Node`, which is a subclass of the OMNeT++ `cSimpleModule` class, and `RouteT`, which is a new class used to handle dynamic route tables. Each node has an input/output gate for each incoming/outgoing edge. Nodes communicate by means of `cMessage`s sent by the source node through one of its output gates and received by the corresponding destination node through one of its input gates. Whenever a node receives a message, it reacts by invoking its `handleMessage()` method. For our purposes, we use two subclasses of `cMessage` (namely, `Interest` and `Data`) to represent control packets and data packets. According to the type of message received, the `handleMessage()` method invokes either `handleInterest()` or `handleData()`.

Notice that OMNeT++ is a communication-oriented event-driven simulator, so that any action needs to be triggered by a message. If a module needs to schedule in the future an action which will not be triggered by an external event, it needs to generate a self message with a proper delay. In the proposed approach, self messages are used by sensor nodes to trigger the generation of data packets at a given rate, and by the sink to trigger the diffusion of a new Interest message at the beginning of each epoch.

`Interest` messages carry three specific properties: `epoch_id`, which denotes the epoch it belongs to, `path_cap`, which represents the capacity of the best path available from the last relaying node to the sink, and `path_hop`, which represents the length of the best path. The pseudo-code of `handleInterest()` is shown in Figure 3. The method compares the current `epoch` (which is a private property of the `Node`) with the `epoch_id` contained in the Interest message. If they are not equal, a new epoch is detected and the `newEpoch()` method is invoked to recompute local node and edge constraints, possibly taking into account capacity debts according to the pseudo-code of Figure 4.

```
Node::newEpoch() {
    // energy
 1  residual_energy += node_power*deltaT - used_energy;
 2  used_energy = 0;

    // bandwidth
 3  for each gate g
 4    if (g->residual_cap*deltaT < g->link_use)
 5      g->residual_cap += g->link_cap - g->link_use/deltaT;
 6    else
 7      g->residual_cap = g->link_cap;
 8    g->link_use = 0;

    // CPU
 9  if (residual_cpu*deltaT < cpu_use)
10    residual_cpu += cpu_cap - cpu_use;
11  else
12    residual_cpu = cpu_cap;
13  cpu_use = 0;

    // link constraint
14  for each gate g
15    g->constraint = min(residual_energy/g->packet_energy,
                          residual_cpu/packet_cpu,
                          g->residual_cap*deltaT);
16    g->constraint /= deltaT;
}
```

Figure 4: Pseudocode of `Node::newEpoch()` method.

```
Interest *RouteT::update(Interest *msg) {
 1  old_best_gate = bestGate(NULL);
 2  g = msg->in_gate;
 3  g->constraint = min(g->constraint, msg->path_cap);
 4  new_best_gate = bestGate(NULL);
    // interest back-propagation
 5  if (!equal(new_best_gate, old_best_gate))
 6    new_msg = new Interest(new_best_gate);
 7  else
 8    new_msg = NULL;
 9  return(new_msg);
}
```

Figure 5: Pseudocode of `RoutT::update()` method.

The `Interest` message is then used to update the local route table by means of method `RouteT::update()`, shown in Figure 5, which possibly returns a new `Interest` message to be used to notify changed best-path conditions. Interest back-propagation is performed by rows 4-6 of `handleInterest()`.

The actual routing policy is implemented by the `bestGate()` method of the `RouteT` class (shown in Figure 6), which returns the best output gate to be used to send data packets according to the greedy routing metric discussed in Section 3. The best gate is the one which leads to the shortest path among those with maximum residual capacity. The incom-

```
Gate *RouteT::bestGate(Gate *in_gate) {
 1  best_gate = NULL;
 2  for each gate g != in_gate
 3    if ((best_gate == NULL) ||
          (best_gate->path_cap < g->path_cap) ||
          ((best_gate->path_cap == g->path_cap) &&
           (best_gate->path_hop > g->path_hop)))
 4      best_gate = g;
 6  return(best_gate);
}
```

Figure 6: Pseudocode of `RouteT::bestGate()` method.

```
Node::handleData(Data *msg) {
 1  in_gate = msg->in_gate;
 2  g = route->bestGate(in_gate);
 3  send(msg,g);
 4  energy_use += g->packet_energy;
 5  cpu_use += packet_cpu;
 6  g->link_use += 1;
}
```

Figure 7: Pseudocode of `Node::handleData()` method.

ing gate is excluded from the choice.

`RouteT::bestGate()` is used both during dissemination and during data collection. During dissemination, it is invoked by `RouteT::update()` before and after updating the routing table, in order to check whether or not the changes need to be back-propagated. During data collection, the method is invoked by `handleData()` to relay data packets toward the sink (as shown in Figure 7).

Whenever a data packet is processed, energy, CPU, and bandwidth usage counters are incremented within `handleData()` in order to be used at the beginning of the subsequent epoch to compute capacity debts (within `newEpoch()`).

5. SIMULATION RESULTS

The simulation model described in Section 4 was used both for validation and for parametric assessment. In order to focus on the inherent properties of the algorithm, simulations were conducted using ideal links with 0 delay and no packet loss, and no heuristic optimizations were applied to the algorithm for reducing the control traffic overhead or speeding-up convergence.

Validation was performed by checking the capability of the self-adapting routing algorithm to handle the maximum sustainable workload, as determined either theoretically (by solving an instance of maxflow whenever applicable) or empirically (by means of iterative simulation runs with decreasing sampling rates) [4]. The self-adapting algorithm always converged to a cycle-stable routing strategy able to sustain the expected maximum sampling rate.

Parametric assessment was performed in order to evaluate the scalability of the proposed approach in terms of convergence speed, routing efficiency, buffering requirements, and control overhead. Since the parameters of interest are strongly dependent on the network topology, on the monitoring task, and on the constraints, a thorough parametric assessment would require a sensitivity analysis conducted on

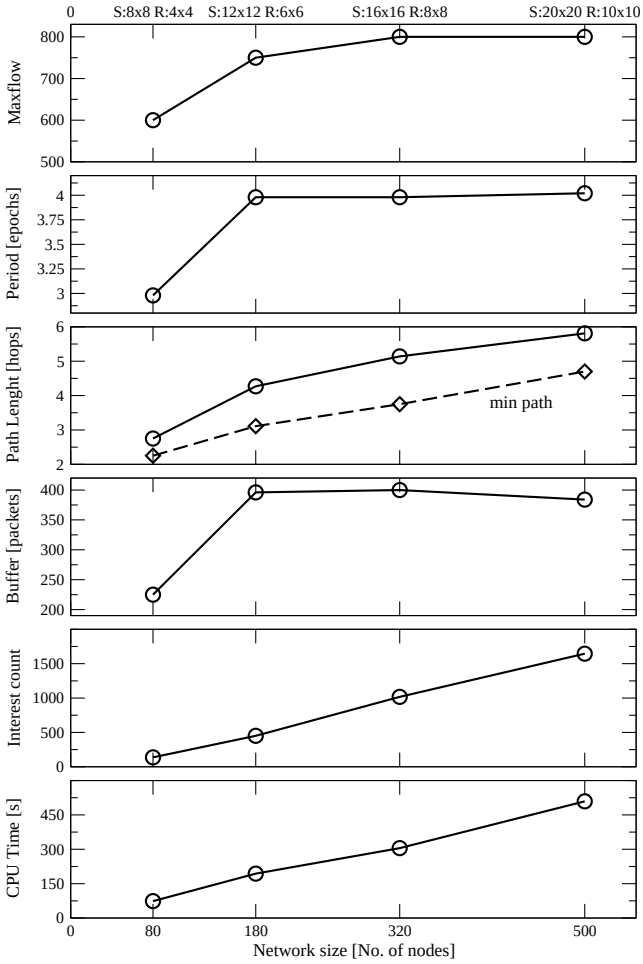

Figure 8: Simulation results.

the results of extensive experiments. Here we only report preliminary results obtained on a family of sensor networks with fixed topology (shown in Figure 1) assigned with the same task (the one called *corner monitoring* in Section 3).

Scalability was tested by increasing the density of the sensors and routers deployed on the square region, while keeping unchanged their relative density and the relative size of the sampled corner. The scalability test was repeated by changing the degree of connectivity, the area of the sampled region, the spread of the power constraints (randomly assigned to each node within a given range), the ratio between node and edge constraints, the relative density of sensors and routers, and the length of the epochs.

Figure 8 plots the results of the scalability test performed on networks dominated by the power constraints imposed to the nodes (200 packets per time unit), with a relative density of 4 sensors per router, with a sampled area of 1/16 of the total area, with epochs of length 1, and with the maximum sustainable sampling rate (according to the pre-computed MSW). The results, plotted as functions of the number of nodes, are discussed in the following.

- *Maxflow* shows the maximum number of packets routed

in a time unit. The value of 600 obtained for the smallest network has been discussed in Section 3. The sustainable maxflow increases with node density because of the larger number of routers at the boundaries of the sampled region (cut A in Figure 1.a). However, the maxflow cannot exceed 800 because of the bottleneck imposed by the four routers directly connected to the sink (cut B in Figure 1.a).

- *Period* is the repetition time of the cycle-stable strategy, automatically evaluated on the Fourier transforms of the time series representing the flow across each edge as a function of the epochs. The shorter the period, the higher the convergence speed and the lower the time variance of the traffic across each edge.
- *Path length* is the average number of hops to the sink, which can be compared to the average minimum path (dashed line in Figure 8) in order to evaluate routing efficiency. On average, the difference between maxflow and minpath is of about 1 hop.
- *Buffer* is the maximum buffer size required to deal with over-allocations, evaluated by looking at the maximum debt observed during the whole simulation run. It is worth noting that buffer requirements do not depend on the size of the network.
- *Interest* is a measure of the control traffic overhead expressed by the total number of interest packets exchanged per epoch, which scales almost linearly with the number of nodes (in the biggest network each node handles on average 3 interest packets per epoch).
- *CPU time* reports the time taken by OMNeT++ to simulate the network for 1,000 epochs on a Centrino 2 Duo T7300 with 2GB of RAM. The CPU time can be used to evaluate the suitability of the simulation model for design exploration.

The remaining experiments (not reported in detail for space limitations) have shown that: i) the length of the epoch affects only the buffer size (which scales linearly) and the ratio between control traffic and data traffic (which is inversely proportional to the epoch length); ii) the spread of the constraints reduces the maximum sustainable workload and increases the average path length; iii) when the sampling rate of the sensors is lower than the maximum sustainable rate both buffering requirements and control traffic overhead decrease; iv) the degree of connectivity impacts the maxflow, the path length, and the control traffic overhead.

5.1 Conclusions and future work

The paper has presented a simulation model of a self-adapting maxflow routing algorithm for wireless sensor networks. The simulation results presented in Section 5 show the effectiveness of the proposed approach and provide valuable data to evaluate its practical applicability.

The current work in this field is focused on three main aspects: evaluating the effects of the non idealities of the wireless links by modeling packet loss and propagation delay, conducting a thorough sensitivity analysis by running systematic parametric experiments, adding heuristic optimization to the routing algorithm to reduce the control traffic and to speed up convergence, and evaluating the adaptation capabilities of the WSN to environmental changes.

6. REFERENCES

[1] OMNeT++ community site: OMNeT++ Discrete Event Simulation System, Last accessed: August 2008. `http://www.omnetpp.org/`.

[2] K. Akkaya and M. Younis. A survey of routing protocols in wireless sensor networks. *Elsevier Ad Hoc Network Journal*, 3(3):325–349, 2005.

[3] R. Amirtharajah, J. Collier, J. Siebert, B. Zhou, and A. Chandrakasan. Dsps for energy harvesting sensors: applications and architectures. *IEEE Pervasive Computing*, 4(3):72–79, 2005.

[4] A. Bogliolo, E. Lattanzi, and A. Acquaviva. Energetic sustainability of environmentally powered wireless sensor networks. In *Proceedings of ACM PE-WASUN*, pages 149–152, 2006.

[5] H.-H. Chen and Y. Yang. Guest editorial: Network coverage and routing schemes for wireless sensor networks. *Elsevier Computer Communications*, 30(14-15):2697–2698, 2007.

[6] F. Dressler. A study of self-organization mechanisms in ad hoc and sensor networks. *Elsevier Computer Communications*, 31:3018–3029, 2008.

[7] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[8] L. C. Klopfenstein, E. Lattanzi, and A. Bogliolo. Implementing energetically sustainable routing algorithms for autonomous wsns. In *Proceedings of WoWMoM*, pages 1–6, 2007.

[9] E. Lattanzi, E. Regini, A. Acquaviva, and A. Bogliolo. Energetic sustainability of routing algorithms for energy-harvesting wireless sensor networks. *Elsevier Computer Communications*, 30(14-15):2976–2986, 2007.

[10] P. Levis, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo. The emergence of a networking primitive in wireless sensor networks. *Communications of the ACM*, 51(7):99–106, 2008.

[11] V. Mhatre and C. Rosenberg. Energy and cost optimizations in wireless sensor networks: A survey. In A. Girard, B. Sanso, and F. Vazquez-Abad, editors, *Performance Evaluation and Planning Methods for the Next Generation Internet*. Kluwer Academic Publishers, 2005.

[12] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of IPSN*, page 64, 2005.

[13] A. Seraghiti and A. Bogliolo. Self-adapting maxflow routing for autonomous wireless sensor networks. In *International Conference on Sensor Technologies and Applications (SENSORCOMM-08)*, 2008.

[14] M. Yarvis and M. Zorzi. Special issue on energy efficient design in wireless ad hoc and sensor networks. *Elsevier Ad Hoc Networks*, 2008.

[15] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Elsevier Computer Networks*, 52:2292–2330, 2008.