

Self-Adapating Maxflow Routing for Autonomous Wireless Sensor Networks

Andrea Seraghiti and Alessandro Bogliolo
Information Science and Technology Institute
University of Urbino
Email: alessandro.bogliolo@uniurb.it

Abstract—Energy-harvesting wireless sensor networks (EH-WSNs) can be modeled as flow networks with channel capacities limited by the ratio between the environmental power available at the nodes and the energy they need to process each packet. The maximum workload that can be sustained by environmental power is the *maxflow* that can be routed across the network from sensor nodes to the sink.

This paper presents a theoretically-optimum maxflow routing algorithm that makes EH-WSNs able to transparently adapt to time-varying environmental power conditions during their normal operation.

I. INTRODUCTION

Battery-operated systems are usually designed in order to maximize their lifetime under the energy constraints imposed by the limited capacity of their batteries [1]. Environmentally-powered systems, on the contrary, should be designed in order to maximize the workload they can sustain using only the limited amount of power they can take from the environment [2], [3]: As long as the power consumption of a system does not exceed the harvested power, it can keep working for an unlimited amount of time.

The paradigm shift from energy-constrained lifetime optimization to power-constrained workload maximization has been recently applied to *energy-harvesting wireless sensor networks* (EH-WSNs) by Bogliolo et al. [4], who have developed a new theoretical framework for modeling and optimizing energetic sustainability. The following results have been achieved within this framework in the last two years:

- 1) EH-WSNs can be represented as flow networks, where bandwidth limits are imposed by the ratio between the environmental power available at each node and the energy required to process and route information packets [4];
- 2) the *maximum energetically-sustainable workload* (MESW) of an EH-WSN subject to given environmental conditions can be determined by solving an instance of *maxflow* [5], [4];
- 3) MESW can be used as a design metric for optimizing the deployment of an EH-WSN under stationary environmental conditions [6];
- 4) given an EH-WSN and the environmental power available at each node, the MESW is a reachable upper bound for the workload that can be actually sustained by any routing algorithm [6];

- 5) the solution of maxflow induces non-deterministic routing tables that can be actually applied at sensor nodes in order to route the theoretical MESW [6], [7];
- 6) the whole sensor network can be viewed as a distributed computer system that can be programmed to solve maxflow in order to compute on the field the optimal routing strategies needed to maximize the sustainable workload [7].

Routing optimality is strongly dependent of environmental power conditions, that usually change over time and are often unknown at design time. The routing strategy optimized for a given power distribution can be counter productive if applied under different conditions [6]

Although the distributed implementation of maxflow [7] allows the sensor network to update its routing tables whenever a significant change of environmental power is detected, maxflow computation entails a non-negligible overhead both in terms of time and energy that could be spent by the network to perform normal operations, thus making it necessary to trade off routing optimality for recomputation frequency.

This paper presents a self-adapting maxflow routing algorithm that makes EH-WSNs able to transparently adapt to the environmental power during their normal operation.

II. THEORETICAL FOUNDATION

A. Maximum Energetically Sustainable Workload

Any network can be modeled as a weighted directed graph $G = (V, E)$ with vertices $v \in V$ representing network nodes and edges $e \in E$ representing links among them: An edge $e_{i,j}$ from node v_i to node v_j exists iff there is a direct link from node i to node j . Each node (say v) of an autonomous WSN can be annotated with the environmental power available at that node $P(v)$. The energy required by node v to process (or generate) a packet and to forward it through one of its outgoing edges (say, e) is expressed by $E(v, e)$. Since the ratio between $P(v)$ and $E(v, e)$ represents an upper bound to the number of packets per time unit that can be processed at node v using only environmental power, power constraints can be expressed as capacity constraints:

$$C(e) = P(v)/E(v, e)$$

and the sensor network can be treated as a *flow network*.

Each node, however, may have multiple outgoing edges that share the same power budget, so that capacity constraints

cannot be independently associated with edges, as in standard flow networks. Rather, local sustainability constraints must be associated with network nodes, according to the following equation:

$$\sum_{e \text{ exiting from } v} F(e)E(v, e) \leq P(v)$$

where $F(e)$ is the flow (i.e., the packet rate) across edge e . For the sake of simplicity, we assume that transmission power is not dynamically adapted to the actual distance between source and destination nodes, so that the energy per packet [8] can be considered as independent from the edge of choice. This simplifying assumption, that holds for many real-world networks, allows us to transform a node-constrained flow network into an equivalent edge-constrained flow network where each original node is split into an input node (destination of all incoming edges) and an output node (source of all outgoing edges), connected by an internal edge with capacity $P(v)/E(v)$. All other edges, representing the actual links among the nodes, are annotated with their channel capacities.

Since the main task of a sensor network consists of collecting information from the sensors, any sensor network can be modeled as a network with multiple source nodes (the sensors) and one destination node (the sink). Networks with multiple sink nodes can be easily modeled as single-sink networks by adding a dummy sink collecting packets at no cost from all the actual sinks [4]. The workload of the network can be expressed as the average number of packets that can be routed from the sensors to the sink in a unit of time. The *maximum energetically-sustainable workload* (MESW) of an autonomous WSN under given environmental conditions is the maxflow of the equivalent edge-constrained flow network described above. Hence, it can be determined by solving an instance of maxflow [4].

In the rest of this paper we address the *optimum maxflow routing* (OMFR) problem under time-varying environmental conditions, which aims at finding a self-adapting routing algorithm for an EH-WSN able to route any theoretically sustainable workload (including the MESW), optimally exploiting the environmental power available at any time.

First, we develop an ideal OMFR algorithm that provides an exact solution to the problem under specific simplifying assumptions, then we introduce non-idealities and we assess the robustness of the algorithm when the simplifying assumptions do not hold.

Since the algorithms outlined hereafter do not depend on the nature of the nodes or on the physical nature of the edge constraints, in the following we refer to a generic edge-constrained flow network representing an EH-WSN according to the model described above. Hence, network nodes may be either actual nodes or dummy nodes inserted for modeling purposes, while edge capacities can represent either bandwidth or power constraints.

```
OMFR(packet)
1 forward packet across besthop
2 return
```

Fig. 1: OMFR algorithm.

B. Ideal OMFR algorithm

The capacity constraints of an edge-constrained flow network (representing either channel bandwidth or energetic-sustainability) can be expressed as the number of bits that can be sent across the link in a time unit. Similarly, the bandwidth required by a flow between two nodes can be expressed as the number of bits sent by the source (or received at the destination) in a time unit.

Given a node s and a path P from s to the sink, the *path capacity* of P is the minimum capacity of the edges along the path, while the *routing capacity* of s is the maxflow between s and the sink.

In order to develop an ideal OMFR algorithm we need to introduce the following simplifying assumptions:

Assumption 1. The flow is split into elementary packets which consume 1 unit of bandwidth.□

Assumption 2. The network handles one packet at the time.□

Assumption 3. Each router has an instantaneous knowledge of the residual capacity of the best path from each outgoing edge (say, e) to the sink ($rescap[e]$).□

An incoming packet is handled at a generic node v by the OMFR algorithm outlined in Fig. 1, which simply chooses the link (denoted by *besthop*) that leads to the best path to destination (i.e., the path with the highest residual capacity). The complexity of the algorithm is hidden behind the computation of *besthop*, which is derived from the residual capacities annotated at each edge, as discussed in Section II-D.

The residual capacity of each link needs to be updated both when a new packet is routed across the link and at the edges of time units used to compute the bandwidth. When a packet is routed across a link the residual capacity of that link is decremented by the size of the packet, while at the edges of time units all residual capacities are reset to their nominal values decremented by the *bandwidth debt* possibly coming from the previous time unit. A bandwidth debt occurs whenever the overall size of all packets routed across a given link in a time unit exceeds the capacity of that link.

Since bandwidth debts violate the physical constraints imposed by edge capacities, their meaning needs to be discussed. Depending on the nature of the constraint (that represents either bandwidth or power limitations) the excess flow that causes a bandwidth debt can be interpreted either as the amount of packets enqueued at some node waiting for the physical resources (bandwidth or energy) needed to process them, or as the extra energy taken at some node from an auxiliary battery that needs to be recharged in the next time unit.

Example 1. Figure 2 exemplifies the behavior of the routing

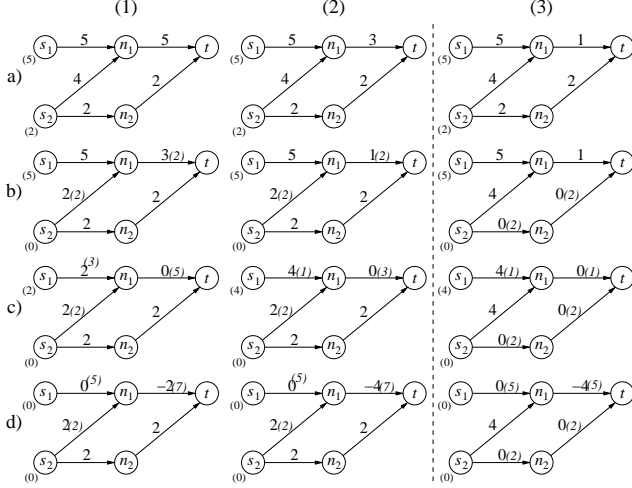


Fig. 2: Example of behavior of routing algorithm OMFR.

algorithm applied to route the packets generated by 2 sensors (namely, s_1 and s_2) across a simple network of 5 nodes, with a common sink t . The first sensor node generates a flow with bandwidth requirement 5, the second sensor generates a flow with bandwidth requirement 2. Residual capacities are annotated above each edge. The Figure shows the status of the network at different stages of the execution of the routing algorithm for three time units. Graphs referring to the same time unit are aligned on the same column.

Figure 2.1.a shows the original edge capacities at the beginning of the first time unit. As a worst case, we assume that within each time unit the packets from source s_2 are generated before those coming from source s_1 . The routing algorithm applied at node s_2 sends both packets to node n_1 since it leads to destination t through a better path (with residual capacity 4 rather than 2). When both packets from s_2 have been delivered, the status of the network becomes 1.b and s_1 starts generating its packets. Since there is a single path from s_1 to t , no routing decisions need to be taken, but it is worth noting that the residual capacity of the edge from n_1 to t is lower than the bandwidth requirement of s_1 . Hence, the first 3 packets can be routed without exceeding network capacity, while the last 2 packets cause a bandwidth debt, denoted in Fig. 2.1.d by a negative residual capacity -2.

At the beginning of time unit 2 (Figure 2.2.a) both flow requirements and edge capacities are reset, but the edge from n_1 to t is annotated with capacity 3 (rather than 5) because of the capacity debt. Then the algorithm proceeds following the steps illustrated in column 2, increasing the bandwidth debt from -2 to -4. As a result, the capacity from n_1 to t at the beginning of time unit 3 is only 1 (Figure 2.3.a). Such a low capacity, however, has the beneficial effect of making the path across node n_1 less convenient than that across node n_2 to route packets from s_2 to t . Hence, the two packets from s_2 now go through the lower path, leaving the upper one to the

flow from s_1 . Since the bandwidth debt at the end of time unit 3 is equal to that obtained at the end of time unit 2, the routing strategy won't change any more. \square

Theorem 1. *Given an EH-WSN G with a sustainable workload, if assumptions 1 to 3 hold, algorithm OMFR converges to a routing strategy which is an exact solution of the OMFR problem.*

Proof. Assume, by contradiction, that the routing algorithm does not converge, so that the residual capacity of (some of) the edges in G keeps decreasing. At the end of a given time unit (say, h) we consider the edge e , from i to j , with the lowest residual capacity $C_{i,j}^{(h)}$. If the routing algorithm does not converge, for each time unit h there is a time unit $k > h$ such that the residual capacity of edge e at the end of k is lower than that at the end of h . In symbols:

$$C_{i,j}^{(k)} < C_{i,j}^{(h)}$$

The decrease of the residual capacity in e from time unit h to time unit k means that the average flow routed across e in that interval exceeds the bandwidth of e . We distinguish two different cases.

In the first case all the point-to-point flows routed across e have no alternative paths to their destinations. Hence, if the sum of the flows exceeds the capacity of e , then the workload is not sustainable for G . A contradiction.

In the second case the edge is also used to route at least a multipath point-to-point flow. Since the routing metric used by the algorithm is based on residual capacities, a path including edge e can be taken iff all the alternatives have lower residual capacities. Since e was the edge with lowest residual capacity at time h , then it can be taken at time k iff the residual capacities of all alternative paths have become lower in the mean time. But this may happen iff the average flow routed across all alternative paths has exceeded their nominal capacities, meaning that the workload is not sustainable for G . A contradiction. \square

C. Non idealities

In this subsection we relax the constraints imposed by Assumptions 1 to 3 and we study the behavior of the algorithm under more realistic conditions. Assumption 1 imposes the network to handle packets of elementary size in order to allow the routing algorithm to take atomic decisions contributing to the globally-optimum routing strategy. Assumption 2 imposes the network to handle one packet at the time in order to make it possible for the algorithm to route each packet by taking into account the effects (in terms of residual capacities) of all the packets routed up to that point. In other words, Assumption 2 gives a precise meaning to the "instantaneous knowledge" assumed by Assumption 3.

The following example shows what happens when Assumption 1 doesn't hold and the routing algorithm deals with packets greater than 1.

Example 2. Fig. 3 shows a simple example of a network with a sensor node (S_2) that generates a flow of bandwidth 4 that

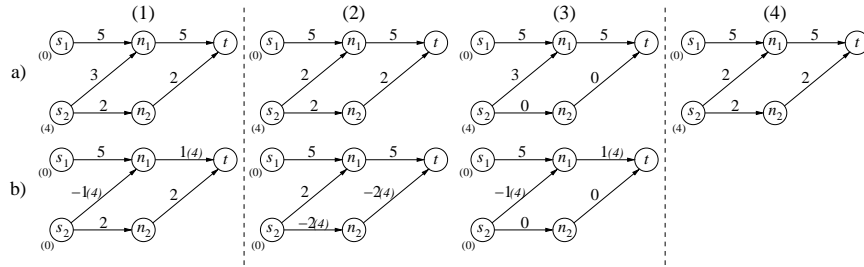


Fig. 3: Behavior of routing algorithm OMFR with packets of size > 1 .

consists of just one packet per time unit of size 4. Although the routing capacity of the network is sufficient to route the flow, the size of the packet exceeds the capacity of all the paths to the sink.

According to algorithm OMFR, in time unit 1 the packet is routed through node n_1 , causing a bandwidth debt of -1 on the edge from s_2 to n_1 . In time unit 2 the upper path has no longer the highest capacity, so that the packet can be routed through node n_2 , causing a bandwidth debt of -2 on both the edges along the path. The debt is compensated in time unit 3, but the upper path exhibits a higher capacity so that it is chosen for routing the packet, leaving the edge from s_2 to n_1 with the same debt occurred at the end of time unit 1.

Since at the beginning of time unit 4 the flow network is in the same conditions encountered at the beginning of time unit 2, OMFR takes the same routing decisions, leading to a cycle-stable routing strategy with a period of 2 time units. \square

The behavior exhibited by the algorithm in Example 2 can be generalized by stating that OMFR converges to a cycle-stable routing strategy that is able to route any sustainable workload. A similar effect can be observed by relaxing the requirements imposed by Assumptions 2 and 3.

Consider, in particular, what happens if the metrics used by the routing algorithm are not based on real-time knowledge of all residual capacities, but on static information updated only at the beginning of each time unit. Since the algorithm doesn't perceive the effect of the routing decisions taken within the current time unit, it keeps routing packets through the same paths which were the most convenient ones for each flow at the beginning of the time unit. In other words, each point-to-point flow is routed across a unique path decided at the beginning of the time unit. The lack of feedback on the effects of routing decisions causes sub-optimal local choices that may lead to bandwidth debts at the end of some time units. The debts accumulated in a time unit, however, are properly taken into account at the beginning of the subsequent time unit, possibly leading to different routing decisions that compensate the previous ones.

On the other hand, if the metrics are not updated instantaneously, then the network can also handle more than one packet at the time without impairing the optimality of the algorithm, so that the second assumption can be relaxed as well.

Fig. 4: BESTPATH routine.

Following the same procedure used to prove Theorem 1, it can be shown that the algorithm converges to a cycle-stable maxflow routing strategy. Periodicity depends on the network, on the constraints, on the workload and on time granularity (i.e., on the recomputation frequency of residual capacities).

D. Metric computation

Two data structures need to be maintained at each node to enable runtime computation of the metrics required by the OMFR algorithm: an array with the residual capacity of each outgoing edge, `linkcap[i]`, and an array with the residual capacity of the best path leading from the current node to the sink through the i -th edge, `rescap[i]`. Index i , representing the outgoing edge, can be replaced by the name (or number) of the node the edge leads to. Two additional variables are used at each node to store the current value of the capacity of the best path to destination (`bestpathcap`) and the index of the edge leading to the best path (`besthop`).

At time 0, all the nodes but the sink are initialized by setting `bestpathcap = 0` and `linkcap[i]` to the overall capacity of the i -th outgoing edge (possibly determined by taking environmental power and packet energy into account). The sink generates a control packet stating that it has unlimited bandwidth and it sends the packet back to all nodes directly connected to it. The packet is backpropagated across the network by the BESTPATH function (Fig. 4) which is executed at each node upon reception of a control packet (`in_packet`).

Consider a node (say, s_1) receiving the control packet from another node (say, s_2). Since the capacity of a path is the minimum of the capacities of the edges it traverses, the residual path capacity from s_1 to the sink through s_2 is computed by the instance of BESTPATH running on node

s_1 as the minimum between the residual capacity of the link leading to s_2 and the residual best-path capacity from s_2 to the sink (first row of the pseudo-code of Fig. 4). The residual path capacity through node s_2 is then compared with the maximum of the residual path capacities available from all other outgoing edges in order to possibly update and back-propagate `bestpathcap` and `besthop` information.

Notice that backpropagation terminates in finite time since the control packets received at each node generate additional packets iff they affect the best path to destination. This inherently avoid loops.

At the end of backpropagation, each node has a complete knowledge of the residual capacity of the best path to the sink across each possible link originating at that node, stored in `rescap[i]`. In addition, each node keeps track of the `bestlink` to each destination (`besthop`), which is the link that leads to the best path to the sink. The reason why `besthop` is maintained in a separate variable, rather than being computed from `rescap[i]`, is that it is more efficient to keep it up to date during backpropagation, and that additional metrics (such as hop minimization) could be taken into account when choosing the best link among those possibly leading to paths with the same bandwidth.

In order to use `BESTPATH` to implement the ideal version of `OMFR`, backpropagation of control packets should be repeated at the beginning of each time unit and then used to notify bandwidth changes at each step of the routing algorithm. To this purpose, whenever a router sends a packet to the next hop, it should compute (and possibly notify) the effect of the packet transmission in terms of residual bandwidth. Consider a packet of size w routed by n across link e , which leads to the best available path to the sink. The transmission of the packet reduces by w , for the current time unit, the residual bestpath capacity from n to the sink across link e , thus requiring both `bestpathcap` and `rescap[e]` to be recomputed. If `bestpathcap` changes, then a control packet needs to be back-propagated to notify the change to all fan-in nodes. Notice that, at subsequent hops, no redundant information about capacity changes is backpropagated. In fact, the control packets possibly generated by other routers encountered by the same packet along its path are dropped by the `BESTPATH` routine running on the routers already visited by the packet.

Although the procedure described above could be implemented in practice, it is of limited practical interest because it causes unacceptable bandwidth and computation overhead.

On the other hand, bandwidth is an average metric and the `OMFR` algorithm provides optimal solutions even if it is based on periodic estimates of the available bandwidth. Hence, residual capacities and bestpath information could be periodically recomputed rather than instantaneously updated.

The periodic recomputation of residual path capacities is performed by means of a backward network traversal triggered by the sink, which generates and sends back a control packet annotated with the *recomputation epoch*, which is an integer number incremented whenever a new recomputation is triggered.

```

NEWPOCH(in_packet)
1 if (in_packet.epoch > epoch)
2   epoch = in_packet.epoch
3   foreach outgoing edge e
4     caplink[e] = cap[e]-debt[e]
5     debt[e] = 0
6   bestpathcap = 0
7   return

```

Fig. 5: `NEWPOCH` routine.

When a node receives a control packet, it first launches the `NEWPOCH` routine (Fig. 5) to compare the current epoch (stored in a variable local to the node) with the `epoch` property of the incoming packet. If `in_packet.epoch` is greater than the local value, then a new epoch is detected and the `caplink[]` array is re-initialized by taking into account the inherent capacity of each edge and the bandwidth debt possibly accumulated in the previous epoch. Then the incoming packet is processed by the `BESTPATH` routine and possibly back-propagated.

III. DISCUSSION AND CONCLUSIONS

The `OMFR` algorithm introduced in Section II has the inherent capability of adapting to time-varying environmental power conditions which are implicitly represented by the capacities (denoted by `cap[e]` in Fig. 5) associated with the edges of the equivalent flow network. As long as the environmental power available at each node is monitored at run time and used at the beginning of each epoch (or time unit) to recompute the energetically sustainable packet rates, the overall routing strategy induced by the `OMFR` algorithm guarantees the best exploitation of the harvested power.

The granularity of the time unit (i.e., the distance between metric recomputation epochs) can be adjusted in order to find a suitable trade off between adaptation speed and computation overhead. Recomputation epochs can be viewed as sampling points for the time-varying distribution of environmental power. The time discretization induced by the periodic recomputation of routing metrics provides an approximate perception of the actual behavior of the power available at each node. However, if the power value $P(v)$ used at node v to recompute the sustainable link capacity is the average value of the power harvested from the environment since last recomputation epoch, time discretization operates just a smoothing of the time-varying behavior with no effects on the long-term optimality of the routing algorithm.

Although the technical aspects of the application of `OMFR` are beyond the scope of this work, it is worth mentioning that both energy buffers (i.e., rechargeable batteries) and packet queues are required at the nodes to compensate short-term variations of environmental power and to provide the required support to the actual implementation of bandwidth debts. The size of buffers depends on the variance of the environmental conditions and on the distance between recomputation epochs.

In conclusion, this paper has introduced a self-adapting optimum maxflow routing algorithm able to exploit envi-

ronmental power to maximize the workload of an energy-harvesting wireless sensor network. Even if the optimality of the algorithm has been theoretically proven, a simulation framework is currently under development to explore the trade-off between adaptation speed and computational overhead, to evaluate the size of energy and packet buffers, and to evaluate the impact of wireless channel models on the behavior of the algorithm.

REFERENCES

- [1] V. Mhatre and C. Rosenberg, "Energy and cost optimizations in wireless sensor networks: A survey," in *Performance Evaluation and Planning Methods for the Next Generation Internet*, A. Girard, B. Sanso, and F. Vazquez-Abad, Eds. Kluwer Academic Publishers, 2005.
- [2] R. Amirtharajah, J. Collier, J. Siebert, B. Zhou, and A. Chandrakasan, "Dsps for energy harvesting sensors: applications and architectures," *IEEE Pervasive Computing*, vol. 4, no. 3, pp. 72–79, 2005.
- [3] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA: IEEE Press, 2005, p. 64.
- [4] A. Bogliolo, E. Lattanzi, and A. Acquaviva, "Energetic sustainability of environmentally powered wireless sensor networks," in *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*. New York, NY, USA: ACM Press, 2006, pp. 149–152.
- [5] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton University Press, 1962.
- [6] E. Lattanzi, E. Regini, A. Acquaviva, and A. Bogliolo, "Energetic sustainability of routing algorithms for energy-harvesting wireless sensor networks," *Elsevier Computer Communications*, vol. 30, no. 14-15, pp. 2976–2986, 1997.
- [7] L. C. Klopfenstein, E. Lattanzi, and A. Bogliolo, "Implementing energetically sustainable routing algorithms for autonomous wsns," in *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, 2007, pp. 1–6.
- [8] A. Y. Wang and C. G. Sodini, "On the energy efficiency of wireless transceivers," in *Proc. of IEEE Conference on Communications*, 2006, pp. 3783–3788.