14 Sensor networks

Wireless sensor systems have a broad range of civil and military applications such as controlling inventory in a warehouse or office complex, monitoring and disseminating traffic conditions, or exploring regions affected by a disaster or military conflict. With recent advances in microelectromechanical systems (MEMS) technology and its associated interfaces, signal processing, and radio and infrared circuitry the focus has shifted away from limited macrosensors communicating with base stations to creating wireless networks of communicating microsensors that aggregate complex data to provide rich, multi-dimensional pictures of the environment. While individual microsensor nodes are not as accurate as their macrosensor counterparts, the networking of a large number of nodes enables high quality sensing networks with the additional advantages of easy deployment and fault-tolerance. These characteristics make microsensors ideal for deployment in otherwise inaccessible environments where maintenance would be inconvenient or impossible.

Communication within a sensor network can be classified into two categories: *application* and *infrastructure*. The network protocol must support both types of communication.

Infrastructure communication refers to the communication needed to configure, maintain, and optimize operation. Thus, infrastructure communication is needed to keep the network functional, ensure robust operation in dynamic environments, as well as optimize overall performance.

Application communication consists of two communication modes: sensor-to-observer and observer-to-sensor communication. Sensor-to-observer communication is the by far dominating mode. Hence, it is of highest priority to find flexible and efficient protocols for this mode. In general, the problem of forwarding information to a single point is called *gathering*. For the rest of this section, we will study how to come up with suitable routing algorithms for gathering. First, we study the problem of gathering in arbitrary adversarial environments, and then we study the problem of gathering in an adversarial environment where the topology is restricted to a line or cycle.

14.1 Gathering in arbitrary adversarial environments

In order to compare our gathering algorithm with a best possible gathering algorithm, we will use competitive analysis.

Suppose that we have an adversary that has full control over the network topology and the injection of packets. Our aim will be to find an online algorithm for this case that can compete with a best possible algorithm. Each time a packet reaches one of its destinations, we count it as one *delivery*. The number of deliveries that is achieved by an algorithm is called its *throughput*. We are interested in maximizing the throughput together with a low storage overhead. In particular, since the adversary is allowed to inject an unbounded number of packets, we allow to drop (or merge) packets so that a high throughput can be achieved with a buffer size that is as small as possible.

Given any sequence of edge activations and packet injections σ, let $\mathrm{OPT}_B(\sigma)$ be the maximum possible throughput (i.e. the maximum number of deliveries) when using a buffer of size B in every node, and let $A_{B'}(\sigma)$ be the throughput achieved by some given online algorithm A with buffer size B'. We call an online algorithm A (t,s)-*competitive* if for all σ and all B, A can guarantee that

$$A_{s\cdot B}(\sigma) \geq t \cdot \mathrm{OPT}_B(\sigma) - r$$

for some $r \geq 0$ that is independent of σ (but may depend on s, B and n). $t \in [0,1]$ denotes here the fraction of the best possible throughput that can be achieved by A and s denotes the space overhead necessary to achieve this. If t can be brought arbitrarily close to 1, A is also called $s(\epsilon)$-*competitive* or simply *competitive*, where $s(\epsilon)$ reflects the relationship between s and ϵ with $t = 1 - \epsilon$. Obviously, it always holds that $s(\epsilon) \geq 1$, and the smaller $s(\epsilon)$, the better is the algorithm A.

In order to perform gathering in an adversarial environment, we use a balancing algorithm. Let $h_{v,t}$ denote the amount of packets in node v at the beginning of time step t. For the observer q, $h_{q,t} = 0$ for all t. In every time unit $t \geq 1$ the T-*balancing algorithm* performs the following operations.

1. For every active edge $e = (v, w)$, check whether $h_{v,t} - h_{w,t} > T$. If so, send a packet from v to w (otherwise do nothing).

2. Receive incoming packets and newly injected packets, and absorb those that reached the observer.

3. Eliminate (or merge) any packet that cannot be stored due to a full buffer.

Recall our competitive model above. We will show that T-balancing algorithm achieves the following result:

Theorem 14.1 ([1]) *Suppose an adversary is allowed to injected an unbounded number of packets and to generate an arbitrary network of maximum degree Δ in each time step. If $T \geq B + 2(\Delta - 1)$, then the T-balancing algorithm is $1 + (1 + (T + \Delta)/B)L/\epsilon$-competitive, where L is the average path length used by successful packets in an optimal solution with buffer size B.*

Proof. Let n be the number of sensor nodes in the system, and let node 0 represent the observer. As mentioned previously, the height of node 0 is always 0, since any packet reaching 0 will be absorbed. For each of the remaining nodes we assume that it has H slots to store packets. The slots are numbered in a consecutive way starting from below with 1. Every slot can store at most one packet. After every step of the balancing algorithm we assume that if a node holds h packets, then its first h slots are occupied. The *height* of a packet is defined as the number of the slot in which it is currently stored. If a new packet is injected, it will obtain the lowest slot that is available after all packets that are moved to that node from another node have been placed.

For every successful packet in an optimal algorithm a schedule can be identified. A schedule $S = (t_0, (e_1, t_1), \ldots, (e_\ell, t_\ell))$ is called *active* at time t if $t_0 \leq t \leq t_\ell$. The *position* of a schedule at time t is the node at which its corresponding packet would be at that time if it is moved according to S. An edge is called a *schedule edge* if it belongs to a schedule of a packet. Suppose that we want to compare the performance of the balancing algorithm with an optimal algorithm that uses a buffer size of B. Then the following fact obviously holds.

Fact 14.2 *At every time step, at most B schedules can have their current position at some node v.*

Next we introduce some further notation. We will distinguish between three kinds of packets: *representatives*, *zombies*, and *losers*. During their lifetime, the packets have to fulfill certain rules. (These rules will be crucial for our analysis. The balancing algorithm, of course, cannot and does not distinguish between these types of packets.) Every injected packet that does not have a schedule will initially be a zombie. Every other packet will initially be a representative. If a packet is injected into a full node, then the highest available loser will be selected to take over its role.

We want to ensure that a representative always stays with its schedule as long as this is possible. Two cases have to be considered for this when the adversary offers a schedule edge $e = (v, w)$:

1. A packet is sent along e: Then we always make sure that this packet is the representative belonging to e.
2. No packet is sent along e: If w has a loser, then the representative exchanges its role with the highest available loser in w. In this case we will also talk about a *virtual* movement. Otherwise, the representative is simply transformed into a loser. In this case, we will disregard the rest of the schedule (i.e. we will not select a representative for it afterwards and the rest of the schedule edges will simply be treated as non-schedule edges).

Furthermore, if a packet is sent along a non-schedule edge $e = (v, w)$, then we always make sure that none of the representatives is moved out of v but only a loser (which always exists if T is large enough).

The three types of packets are stored in the slots in a particular order. The lowest slots are always occupied by the losers, followed by the zombies and finally the representatives. Every zombie that happens to be placed in a slot of height at most $H - B$ will be immediately converted into a loser. Together with Fact 14.2 these rules immediately imply the following fact.

Fact 14.3 *At any time, the number of zombies and representatives stored in a node is at most B.*

Let $h_{v,t}$ be the *height* of node v (i.e. the number of packets stored in it) at the beginning of time step t, and let $h'_{v,t}$ be its height when considering only the losers. The *potential* of node v at step t is defined as $\phi_{v,t} = \sum_{j=1}^{h'_{v,t}} j = \binom{h'_{v,t}+1}{2}$ and the potential of the system at step t is defined as $\Phi_t = \sum_v \phi_{v,t}$. First, we study how the potential can change in a single step. Since no two packets can cross the same edge at the same time, we arrive at the following fact.

Fact 14.4 *At any time t, any edge activated at time t belongs to at most one schedule.*

Hence, an edge can only belong to one or no schedule. To simplify the consideration of these two cases, we consider the edges that are active in a time step one by one, starting with non-schedule edges and always assuming the worst case concerning previously considered edges. When processing these edges, we always use the (worst case) rule that if a loser is moved to some node w, it will for the moment be put on top of all old packets in w. This will simplify the consideration of edges that belong to a schedule. At the end, we then move all losers down to fulfill the ordering condition for the representatives, zombies, and losers. This will certainly only decrease the potential. Using this strategy, we can show the following result.

Lemma 14.5 *If $T \geq B + 2(\Delta - 1)$, then any non-schedule edge does not increase the potential of the system.*

Proof. Consider any fixed non-schedule edge $e = (v, w)$. If no packet is sent along e, the lemma is certainly true. Otherwise, it holds that $h_{v,t} - h_{w,t} > T$. If $T \geq B + 2(\Delta - 1)$, then even after $\Delta - 1$ removals of packets from v and the arrival of $\Delta - 1$ packets at w, there are still losers left in v, and the height of the highest of these is higher than the height of w. Hence, we can avoid moving any representative away from the position of its schedule and instead move a loser from v to w without increasing the potential. □

For schedule edges, only a slight increase in the potential is caused.

Lemma 14.6 *If $T \geq B + 2(\Delta - 1)$, then every schedule edge increases the potential of the system by at most $T + B + \Delta$.*

Proof. Consider some fixed schedule edge $e = (v, w)$. If e is selected for the transmission of a packet, then we can send the corresponding representative along e, which has no effect on the potential.

Otherwise, it must hold that $\delta_e = h_{v,t} - h_{w,t} \leq T$. In this case, the representative R for e has to be moved virtually or transformed into a loser.

First of all, note that our rule of placing new losers on top of the old packets makes sure that the height of the representative in v does not increase. Furthermore, there are two ways for w to lose losers before considering e: either an unused schedule edge to w forced a virtual movement of a representative to w, or a used non-schedule edge from w forced to move a loser out of w. Let s be the number of edges with the former property and ℓ be the number of edges with the latter property. If w had r representatives (and zombies) at the beginning of t, then it must hold that $r + s - (\Delta - \ell) \leq B$ to ensure that at the end of step t w has at most B representatives. Thus, $r + s + \ell \leq B + \Delta$. Hence, if there is still a loser left in w when considering e, the highest of these must have a height of at least $h_{w,t} - (B + \Delta)$. Therefore, if $h_{w,t} > B + \Delta$, then it is possible to exchange places between R and a loser in w so that the potential increases by at most

$$h_{v,t} - (h_{w,t} - (B + \Delta)) = \delta_e + B + \Delta \ .$$

Since $\delta_e \leq T$, this is at most $T + B + \Delta$.

If $h_{w,t} \leq B + \Delta$, then it may be necessary to convert R into a loser. However, since $h_{v,t} - h_{w,t} \leq T$, this increases the potential also by at most $T + B + \Delta$. □

In addition to edges, also injection events and the transformation of a zombie into a loser can influence the potential. This will be considered in the next two lemmata.

Lemma 14.7 *Every deletion of a newly injected packet decreases the potential by at least $H - B$.*

Proof. According to Fact 14.3, the highest available loser in a full node must have a height of at least $H - B$. Since the deletion of a newly injected packet causes this loser to be transformed into a representative or zombie, this decreases the potential by at least $H - B$. (Note that in case of a zombie, it might be directly afterwards converted back into a loser, but this will be considered in the next lemma.) □

If an injected packet is not deleted, this will initially not affect the potential, since it will either become a representative or a zombie. However, a zombie may be converted into a loser.

Lemma 14.8 *Every zombie can increase the potential by at most $H - B$.*

Proof. Note that zombies do not count for the potential. Hence, the only time when a zombie influences the potential is the time when it is transformed into a loser. Since we allow this only to happen if the height of a zombie is at most $H - B$, the lemma follows. □

Now we are ready to prove an upper bound on the number of packets that are deleted by the balancing algorithm.

Lemma 14.9 *Let σ be an arbitrary sequence of edge activations and packet injections. Suppose that in an optimal strategy, s of the injected packets have schedules and the other z packets do not. Let L be the average length of the schedules. If $H \geq B + 2(\Delta - 1)$, then the number of packets that are deleted by the balancing algorithm is at most*

$$s \cdot \frac{L(T + B + \Delta)}{H - B} + z \ .$$

Proof. First of all, note that only newly injected packets get deleted. Let p denote the number of schedule edges and d denote the number of packets that are deleted by the balancing algorithm. Since

- due to Lemma 14.5 non-schedule edges do not increase the potential,
- due to Lemma 14.6 every schedule edge increases the potential by at most $T + B + \Delta$,
- due to Lemma 14.7 every deletion of a newly injected packet decreases the potential by at least $H - B$, and
- due to Lemma 14.8 every zombie increases the potential by at most $H - B$,

it holds for the potential Φ after executing σ that

$$\Phi \leq p \cdot (T + B + \Delta) + z \cdot (H - B) - d \cdot (H - B) \ .$$

Since on the other hand $\Phi \geq 0$, it follows that

$$d \leq \frac{p \cdot (T + B + \Delta)}{H - B} + z \ .$$

Using in this inequality the fact that the average number of edges used by successful packets is at most L, and therefore the number of injected packets with a schedule, s, satisfies $s \geq p/L$, concludes the proof of the lemma. □

From Lemma 14.9 it follows that the number of packets that are successfully delivered to their destination by the balancing algorithm must be at least

$$s + z - \left(s \cdot \frac{L(T + B + \Delta)}{H - B} + z \right) - H \cdot n = s \cdot \left(1 - \frac{L(T + B + \Delta)}{H - B} \right) - H \cdot n \ ,$$

where n is the number of sensor nodes. For $H \geq L(T + B + \Delta)/\epsilon + B$ this is at least

$$(1 - \epsilon)s - r$$

for some value r independent of the number of packets successful in an optimal schedule. □

14.2 Gathering in adversarial lines and cycles

Also here we assume that we have a network of unpredictable edges and packet injections, but now we restrict the adversary to choose edges only from a limited set of edges (instead of all possible edges as above). This represents the case where sensors do not move around and only have a limited transmission radius, so that there is only a limited number of other sensors they can communicate with.

Since sensor may frequently be in sleep mode to save energy or there may be temporary obstacles, not all of these sensors may be available at any time. We use the following formal model for this case:

We assume that we have a network $G = (V, E)$ of static topology but unreliable edges. All injected packets have the same destination in G. We assume that time proceeds in synchronous steps. Every edge that is active at some step can transport at most one packet, and every packet needs one step to cross an edge. The state of the edges and the packet injections are under adversarial control.

Given nodes with buffer size B, we allow an adversary to inject an arbitrary number of packets and to activate (or deactivate) and arbitrary set of edges in E in each time step as long as no packet ever has to be deleted when using an optimal routing algorithm. That is, every node only has to hold B packets in transit at any time, and the other packet have been successfully delivered to the destination. For every successful packet, a *schedule* (i.e. a sequence of movements across active edges over time) can be specified to reach its destination. Recall that the number of packet deliveries achieved by an algorithm is called its *throughput*.

In order to compare the performance of a best possible strategy with our online strategies, we will use competitive analysis. We call an online algorithm A c-competitive if for all sequences of edge activations and packet injections for which an optimal algorithm with buffer size B never has to delete a packet, also A with buffer size $c \cdot B$ never has to delete a packet (which implies that it asymptotically achieves the same throughput as the optimal algorithm). Certainly, $c \geq 1$ for any algorithm A, and for an online algorithm to be useful and scalable for sensor networks, c should be as small as possible.

14.3 Line graph

Consider the line graph with nodes numbered from 1 to n, where n is the destination node. From the analysis of the T-balancing algorithm above we know that it can be at best $\Theta(n)$-competitive. So the question is whether there is an online algorithm that can do better than that. We start with a lower bound that proves that for any online routing algorithm there is an adversary that can force the algorithm to use a buffer size of $\Omega(B \log n)$ at at least one node to avoid dropping a packet, whereas the optimal routing algorithm uses a buffer of size B.

Theorem 14.10 ([2]) *Any deterministic online algorithm is $\Omega(\log n)$ competitive on a line graph of n nodes.*

Proof. To simplify the proof we use an adaptive adversary. However, the result also holds for oblivious adversaries. We consider the case where $B = 1$. This means that in the optimal routing algorithm there is no more than one packet at any node at any given time. Also, assume that $n' = n - 1$ is a power of 2. In the following the nodes in an interval $[a, b]$ means all the nodes numbered from a to b (i.e. including a and b).

The adversary works in rounds. In round 1, the adversary injects one packet at each of the nodes in $[1, \frac{n'}{2}]$. Then the adversary activates all edges (i, j) such that $j = i + 1$ and $1 \leq i \leq \frac{n'}{2}$ for n' time steps. Thus the packet injected at node numbered i, $1 \leq i \leq n'/2$, could be moved to node numbered $\frac{n'}{2} + i$. It holds that either the nodes in $[1, n'/2]$ have at least $n'/4$ of the packets or the nodes in $[\frac{n'}{2} + 1, n']$ have at least $n'/4$ of the packets. Without loss of generality, we shall assume that the nodes in $[\frac{n'}{2} + 1, n']$ have more than $n'/4$ packets. Hence, the average number of packets per node is 0.5. However, in this case, the optimal algorithm will keep all the packets in the interval $[1, \frac{n'}{2} - 1]$. Thus none of the packets in the algorithm in nodes $[\frac{n'}{2} + 1, n']$ are on schedule.

In round 2, the adversary works with the node set ranging from $n'/2+1$ to n' by first placing one packet at $n'/4$ of the nodes in the interval $[n'/2+1, 3n'/4]$. Now the adversary activates edges in a way that a packet in node numbered $\frac{n'}{2}+i$ could be moved to node numbered $\frac{3n'}{4}+i$, $1 \leq i \leq \frac{n'}{4}-1$. It now holds that either the nodes in the interval $[\frac{n'}{2}+1, \frac{3n'}{4}-1]$ or $[\frac{3n'}{4}+1, n']$ have at least half the total packets. W.l.o.g., we shall assume that the nodes in the interval $[\frac{3n'}{4}+1, n']$ have at least half of the total packets. Again the adaptive adversary ensures that all the packets in this interval are not on schedule by using a similar strategy as in round 1. The average number of packets per node in this interval is 1.

The above procedure continues with round k starting with a set of $\frac{n'}{2^{k-1}}$ nodes having at least $\frac{(k-1)\cdot n'}{2^k}$ packets. In round k the adversary injects an additional $\frac{n'}{2^k}$ packets and offers a corresponding number of edges. At the end of the round k, at least $\frac{k \cdot n}{2^{k+1}}$ packets are in $\frac{n'}{2^k}$ nodes. Thus, at least one node has a height of at least $k/2$.

Since after each round the number of nodes that the adversary works with reduces by a factor of 2 there can be at most $\log n'$ such rounds at the end of which there is a node of height at least $\frac{\log n'}{2}$. Note that during this whole procedure, the optimal algorithm has only one packet at any node. If at the end of any round there is a node of height more than $\frac{\log n'}{2}$ then the adversary can stop after that round. $\square$

Using similar arguments, one can also show that for randomized online routing algorithms the adversary creates with a probability of success of at least $1/n$ a node with a height of at least $B \log n$.

Now we come to an online algorithm that can achieve a competitiveness of $O(\log n)$. Before presenting the algorithm, we introduce some notation.

Each node has a buffer to store packets and each buffer has slots numbered consecutively starting from 1. Each slot can store one packet. A *layer* is a consecutive set of $B+1$ slots. The layers are numbered consecutively starting from 0. The slots 1 to $B+1$ belong to layer 0 and $i \cdot (B+1)+1$ to $(i+1)\cdot(B+1)$ belong to layer i, for $i > 0$. The direction of the layer is the direction in which packets can be moved by the algorithm in that layer. Thus, in a Always-Go-Right layer packets can only move to the right and in a Always-Go-Left layer packets can only move to the left. The algorithm maintains layers that are alternating in direction starting with layer 0 being a Always-Go-Right layer. Thus, all even numbered layers are Always-Go-Right layers and all odd numbered layers are Always-Go-Left layers.

Now we are ready to present the *aggressive B-balancing algorithm*, which works as follows in every time step:

1. For every active edge $e=(v,w)$, check whether there exists a layer ℓ with the same direction as e so that v has at least 1 packet in ℓ and w has at most B packets in ℓ. If so, send a packet from v to w (otherwise do nothing).

2. Receive incoming packets and newly injected packets, and absorb those that reached the observer.

The approach of giving layers directions ensures that at most one packet can reach a node for each layer. Hence, if the condition in (1) is satisfied, then the sent packet never has to be stored in a layer higher than its current layer, which results in the following fact.

Fact 14.11 *Starting from the time a packet P was injected, the number of the layer which P belongs to cannot increase.*

For the analysis of the algorithm, some notation is needed.

Definition 14.12 (Position) *The current position (i.e. node) of a packet P in the algorithm is denoted by POS$_{\text{ALG}}(P)$. The current position of a packet P in the optimal algorithm is denoted by POS$_{\text{OPT}}(P)$.*

When there is no confusion about whether a packet P is a packet in the optimal algorithm or the online algorithm, we simply refer to its position as $POS(P)$. The height of a packet and the height of a node are defined below.

Definition 14.13 (Height of a packet) *The height of a packet P in the algorithm, denoted by $h(P)$, is the number of the slot at which it is currently stored.*

Definition 14.14 (Height of a node) *The height of a node u in the algorithm, denoted by $h(u)$, is the highest slot number at which a packet is currently stored at u.*

Definition 14.15 (ALG$_{\geq i}$) *ALG$_{\geq i}$ is the set of all packets P in the algorithm such that $h(P) > i \cdot (B+1)$. We also denote $|ALG_{\geq i}|$ by $n_{\geq i}$.*

We also denote the set of packets in the optimal algorithm as OPT. A monotonic mapping is defined as follows.

Definition 14.16 (Monotonic Mapping) *For two packet sets $\mathcal{P}, \mathcal{P}'$, a mapping $f : \mathcal{P} \to \mathcal{P}'$ is called* left monotonic *(resp.* right monotonic*) if for all $P \in \mathcal{P}$, POS(P) is to the* left*(*right*) of or the same as POS$(f(P))$. When the direction of monotonicity is clear from the context we simply say that f is monotonic.*

One can maintain certain mappings from packets in the algorithm to packets in the optimal algorithm, and the type of mappings we are considering is defined as follows.

Definition 14.17 ($f_{\geq i}$, OPT$_{\geq i}$) *$f_{\geq i}$ is a mapping from ALG$_{\geq i}$ to OPT that is monotonic in the direction of layer i. We define OPT$_{\geq i} = \{P' \in$ OPT : for some $P \in$ ALG$_{\geq i}, f_{\geq i}(P) = P'\}$. Thus we can think of $f_{\geq i}$ as a mapping from ALG$_{\geq i}$ to OPT$_{\geq i}$.*

Now, the following invariants can be maintained after any step (see [2] for a proof):

Invariant 14.18 *At any time, the aggressive balancing algorithm satisfies*

1. *Stability: The number of packets in the system is at most $n \cdot B$,*
2. *Monotonicity: For all $i \geq 0$, $f_{\geq i}$ is injective and right monotonic if i is even and otherwise left monotonic, and*
3. *Containment: For all $i \geq 0$, OPT$_{\geq i+1} \subseteq$ OPT$_{\geq i}$.*

Using this invariant, one can show the following result.

Theorem 14.19 ([2]) *The aggressive B-balancing algorithm is $O(\log n)$-competitive for the line graph.*

This result matches the lower bound in Theorem 14.10, and hence the aggressive balancing algorithm is basically the best one can use in general in a line graph.

14.4 Cycle graph

In the cycle graph, the nodes form a cycle, where one of the nodes is the observer.

If we replace the Always-Go-Right and Always-Go-Left rules by Always-Go-Clockwise and Always-Go-Counterclockwise, then the proof of Theorem 14.19 can be extended to show the following result.

Theorem 14.20 ([2]) *The aggressive B-balancing algorithm is $O(\log n)$-competitive for the cycle graph.*

References

[1] B. Awerbuch, A. Brinkmann, and C. Scheideler. Anycasting in adversarial systems. In *Proc. of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2003.

[2] K. Kothapalli and C. Scheideler. Information gathering in adversarial systems: Lines and cycles. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2003.