Information Gathering in Adversarial Systems: Lines and Cycles

Kishore Kothapalli
kishore@cs.jhu.edu

Christian Scheideler
scheideler@cs.jhu.edu

Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA

ABSTRACT

In this paper we consider the problem of routing packets to a single destination in a dynamically changing network, where both the network and the packet injections are under adversarial control. Routing packets to a single destination is also known as information gathering. Information gathering is an important communication primitive for sensor networks. Since sensor networks have a wide range of civilian and military applications, they have recently attracted a great deal of research attention. Several communication protocols have already been suggested for sensor networks, but not much theoretical work has been done so far in this area. Information gathering is an important primitive to allow an observer to collect information from the sensors. Because sensors usually do not move, they form a static topology of possible communication links, but since sensors may frequently be in sleep mode or their communication may be disrupted by interference or obstacles, communication links may be up and down in an unpredictable way. In this paper, we consider sensor networks forming lines or cycles of unreliable edges. Already these seemingly simple topologies are difficult to handle by online algorithms, and the best previously known algorithms require by a factor of $\Theta(n)$ more buffer size to achieve the same throughput as optimal routing algorithms, where n is the size of the network. We improve this factor to $O(\log n)$ and prove a matching lower bound that holds for *all* online algorithms.

Categories and Subject Descriptors

F.2.8 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Network problems*

General Terms

Algorithms, Reliability, Theory

Keywords

Sensor networks, routing, online algorithms

1. INTRODUCTION

Information gathering is an important communication primitive for sensor networks. The potential for collaborative, robust networks of microsensors has recently attracted a great deal of research attention. Several projects have been initiated in recent years to investigate a variety of communications research aspects in the area of sensor networks [1, 4, 2, 3, 19]. Also, several protocols for determining energy-efficient routes and for aggregating and sending information to an observer along these routes have already been presented and experimentally evaluated [16, 20, 17, 18, 21, 24], but not much theoretical work has been done in this area yet. However, especially for sensor networks, where human intervention is very limited (because sensors may be in hostile territory or concrete walls), theoretical work is important to make sure that the communication algorithms are highly robust, i.e. they work correctly and efficiently under any circumstances. In order to ensure correctness, it is important to use communication methods in sensor networks that are extremely simple (to allow a verification of their correct implementation), and in order to ensure a high efficiency, these methods must be able to adapt themselves to changes in the network so that they use the available resources as efficiently as possible.

For sensor networks, energy is a much more critical factor than in conventional wireless networks because once sensors have been delivered, it may be very difficult, if not impossible, to access and recharge them. Communication is the major consumer of energy in sensors [22]. Wireless interfaces consume energy even if they are just listening for signals from other sensors. Hence, to keep energy consumption at a minimum, a common strategy for sensor networks is to use TDMA (time division multiple access) [25, 17]. Time slots for transmission are chosen among the sensors so that there is no interference with others, and in between these time slots the wireless interface is switched to sleep mode.

Taking this into account, we will model the infrastructure of a sensor network as a directed graph $G = (V, E)$ with a set of nodes V representing the sensors and a set of edges $E \subseteq V \times V$ representing the communication links that can potentially be established between two sensors via radio or infrared signals. Since sensors usually do not move, G has a static topology. However, edges may be unreliable because sensors may be in sleep mode to conserve energy, obstacles or interference with other devices may temporarily obstruct their signals, or they may simply fail. To make sure that our communication algorithms work well under *any* circumstances, we will assume that the state of the edges is under adversarial control.

Protocols for sensor networks have to provide 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 it. A simple form of sensor-to-observer communication is called *information gathering*. In information gathering, there is a single observer, and all messages injected into the network have to be forwarded to it. The aim of this paper is to present a simple, distributed protocol for information gathering that can compete well with best possible protocols with regard to throughput and buffer size. We will concentrate in this paper on two types of topologies for G: lines and cycles. As will become apparent later on in this paper, already these types of networks present serious challenges for keeping the buffer size low (at most logarithmic in the size of the network) while achieving a high throughput.

1.1 Model

As mentioned above, 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 active edge can transport at most one packet per time unit, and every packet needs one time unit to cross an edge. 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) an arbitrary set of edges 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 packets have been successfully delivered to the destination. For every successful packet, a *schedule* (*i.e.* a sequence of movements across edges over time) can be specified to reach its destination. 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, A with buffer size $c \cdot B$ also never has to delete a packet (which implies that it asymptotically achieves the same throughput as the optimal algorithm). Certainly, $c \geq 1$, and for an online algorithm to be useful and scalable for sensor networks, c should be as small as possible.

In the following, B will always mean the buffer size of an optimal routing algorithm.

1.2 Previous results

The study of adversarial models in communication networks was initiated, in the context of queueing disciplines, by Borodin et al. [12]. Other work on adversarial queueing includes [6, 13, 14, 15, 23, 26]. In these papers it is assumed that the adversary has to provide a path for every injected packet and reveals these paths to the system. The paths have to be selected in a way that they do not overload the system. Hence, it only remains to find the right queueing discipline (such as furthest-to-go) to ensure that all of the packets can eventually reach their destinations.

The study of adversarial models was initiated, in the context of routing, by Awerbuch, Mansour and Shavit [11] and further refined by [5, 8, 9, 10, 14]. In these papers the model is used that the adversary does not reveal the paths to the system, and therefore the routing protocol has to figure out paths for the packets by itself. Based on work by Awerbuch and Leighton [10], Aiello et al. [5] showed that there is a simple distributed routing protocol that keeps the number of packets in transit bounded in a dynamic network if, roughly speaking, in each window of time the paths selected for the injected packets require a capacity that is below what the available network capacities can handle in the same window of time.

Also the special case of information gathering (i.e. all packets have the same destination) has already been studied under adversarial routing models. The result by Aiello et al. [5] implies that information gathering (in a somewhat weaker adversarial model than considered here) can be done with competitive ratio $O(n^2)$. Awerbuch et al. [8] consider an adversary similar in power to our adversarial model and improved the competitive ratio to $\Theta(n)$. A similar result was also shown subsequently (with a simpler proof) by Anshelevich, Kempe, and Kleinberg [7]. For general network topologies (i.e. the adversary can interconnect any pair of nodes), a bound of $\Theta(n)$ seems to be best possible, but it was left as an open problem whether there are algorithms for specific topologies that achieve a much better competitive ratio.

1.3 Our results

We show that for lines and cycles there is a simple, deterministic local-control routing algorithm that achieves a competitive ratio of $O(\log n)$. We also present a matching lower bound showing that *any* online algorithm that never has to drop a packet must have a competitive ratio of $\Omega(\log n)$. Although lines and cycles seem to be pretty simple topologies, the adversary can still make the life for a routing protocol very hard. For example, an optimal strategy for a given adversary may move packets back and forth to free up space in buffers into which the adversary will inject packets in the future. Since edges are only active at certain, adversarially selected time steps, packets mistakenly moved by an online algorithm into one direction may be hard to get back. Nevertheless, we show that good online algorithms can be found for lines and cycles.

Apart from achieving a significantly better competitive ratio than the $\Theta(n)$ ratio known before, we note that we use an algorithmic approach that is different from previous approaches in the adversarial routing literature, and we use an analysis different from standard analyses such as potential methods. Hence, both our algorithm and its analysis may be of independent interest.

1.4 Organization of the paper

In Section 2 we prove matching upper and lower bounds for the line graph, and in Section 3 we prove matching upper and lower bounds for the cycle. The paper ends with a conclusion and some open problems.

2. LINE GRAPH

Consider the line graph with nodes numbered from 1 to n, node n being the destination node. In this section, we first present a lower bound on the buffer size used by any deterministic online routing algorithm to perform gathering in the line graph. Later, we propose an online algorithm that achieves an upper bound matching the lower bound up to a constant factor.

2.1 Universal Lower Bound

In this section, we present a lower bound argument that proves that for any online routing algorithm there is an adversary that can force the algorithm to use a buffer of size $\Omega(\log n \cdot B)$ at at least one node whereas the optimal routing algorithm uses a buffer of size B.

THEOREM 2.1. *Any deterministic online routing 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. $\square$

It can also be shown that for randomized online routing algorithms the adversary creates with a probability of success at least $\frac{1}{n}$ a node with a height of $\log n \cdot B$.

2.2 Algorithm

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 in 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.

The position of a packet is defined as follows.

DEFINITION 2.2 (POSITION). *The current position (i.e. node) of a packet P in the algorithm is denoted by $POS_{\mathrm{ALG}}(P)$. The current position of a packet P in the optimal algorithm is denoted by $POS_{\mathrm{OPT}}(P)$.*

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 2.3 (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 2.4 (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 2.5 ($\mathrm{ALG}_{\geq i}$). *$ALG_{\geq i}$ 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 2.6 (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.*

In the analysis, we provide a mapping from packets in the algorithm to packets in the optimal algorithm and the mapping is defined as follows.

DEFINITION 2.7 ($f_{\geq i}, \mathrm{OPT}_{\geq i}$). *$f_{\geq i}$ is a mapping from $ALG_{\geq i}$ to OPT that is monotonic in the direction of layer i. We define $OPT_{\geq i} = \{P' \in OPT : \text{for some} P \in ALG_{\geq i}, f_{\geq i}(P) = P'\}$. Thus we can think of $f_{\geq i}$ as a mapping from $ALG_{\geq i}$ to $OPT_{\geq i}$.*

Now we present Algorithm A for performing routing in the line graph.

Algorithm A

1. For every time step and every active edge (u, v)
2. if there exists a layer l with the direction same as the direction of (u, v) so that u has at least 1 packet in l and v has at most B packets in l then
3. move a packet in layer l from u to v
4. For every time step at every node u
5. Accept incoming packets and injected packets and store them at lowest empty slots available currently.

End Algorithm

Thus, in algorithm A not only the highest packet available but also a packet from any of the layers below may be moved from node u to v. It can be observed that only allowing to move the highest available packet may even result in algorithm A having an unbounded number of packets. It is the ability of algorithm A to treat all packets as "movable" that results in a significant decrease in buffer size requirements. As stated in the following fact, packets however cannot move from a layer l to a layer $l' > l$.

FACT 2.8. *Starting from the time a packet P was injected, the number of the layer to which P belongs to cannot increase.*

2.3 Proof

In this section, we show that algorithm A has a competitive ratio of $O(\log n)$. This is achieved by first arriving at an upper bound on the number of packets in the system at any time. Then we define a legal state for the system and show that the state of the system will stay legal for every time step. Using the legal state, we then show that for nodes to have packets in higher layers there must be many more nodes with packets in the lower layers. This allows to bound the maximum buffer space used by the algorithm.

In the following lemma we first derive an upper bound on the number of packets in the system at any given time.

LEMMA 2.9. *The total number of packets in the system at any time is at most $n \cdot B$.*

PROOF. The proof is done by showing that at every time step, there is an injective and a right monotonic mapping $f_{\geq 0}$ from $\text{ALG}_{\geq 0}$ to $\text{OPT}_{\geq 0}$. At $t = 0$, the mapping exists trivially. Let $f_{\geq 0}$ exist till time t. We consider all actions that the algorithm can perform during time $t+1$. Let us look at active edges at time $t+1$ one at a time and observe the actions of the algorithm and the optimal algorithm along the same active edge. Let us call the active edge as $u \to v$.

- Both the algorithm and optimal algorithm move a packet: Let the algorithm move packet P_a while the optimal algorithm moved packet P_o. If u is to the right of v, then $f_{\geq 0}(P_a)$ is still monotonic. But if a packet P' exists in the algorithm with $\text{POS}(P') = u$ and $f_{\geq 0}(P') = P_o$, by the movement of P_o to v the monotonicity is violated for P'. The monotonicity can be restored by swapping $f_{\geq 0}(P_a)$ and $f_{\geq 0}(P')$. On the other hand, if u is to the left of v, the movement of P_a might violate the monotonicity for P_a. Since the algorithm moved a packet from u to the left of u it implies that $h(u) > B+1$ at time t. Thus, there exists a packet P' with $\text{POS}(P') = u$ and $\text{POS}(f_{\geq 0}(P')) \neq u$. We can hence swap $f_{\geq 0}(P_a)$ and $f_{\geq 0}(P')$.
- Only the algorithm moves a packet: In this case, if the movement is to the right, then the mapping is monotonic after time $t+1$. But if the algorithm moved a packet P_a to the left of its current position, then it follows that $h(u) > B+1$ at time t and as in the previous case, we can swap $f_{\geq 0}(P_a)$ and $f_{\geq 0}(P')$ where $\text{POS}(P') = u$ and $\text{POS}(f_{\geq 0}(P')) \neq u$.
- Only the optimal algorithm moves a packet: If the movement of the optimal algorithm is to the left, then monotonicity would still hold after time $t+1$. However, if the optimal algorithm moved to the right then monotonicity could be violated for the packet P in the algorithm with $\text{POS}(P) = u$ and $f_{\geq 0}(P) = P_o$. In this case, the algorithm failed to move any packet because there did not exist a layer l from which to move the packet. This implies that $h(v) \geq B+1$. Since only B packets in the algorithm that are at v can have their optimal packet also at v, we can find a packet P' with $\text{POS}(P') = v$ and $\text{POS}(f_{\geq 0}(P')) \neq v$. We then swap $f_{\geq 0}(P)$ and $f_{\geq 0}(P')$ thereby restoring monotonicity.

If during time $t+1$ the adversary injects a packet P at node u, we set $f_{\geq 0}(P) = P$. This always ensures monotonicity of $f_{\geq 0}$ and also that $f_{\geq 0}$ is injective since the mapping is provided for all packets in $\text{ALG}_{\geq 0}$. To obtain the upper bound stated in the lemma, we observe that the optimal algorithm can store at most $n \cdot B$ packets at any time. □

From Lemma 2.9, we obtain that $n_{\geq 0} \leq n \cdot B$. We now define a legal state as follows.

DEFINITION 2.10 (LEGAL STATE). *A system is said to be in a legal state if it satisfies*

1. *Monotonicity: For all $i \geq 0$, $f_{\geq i}$ is injective and monotonic, and*
2. *Subset: For all $i \geq 0$, $OPT_{\geq i+1} \subseteq OPT_{\geq i}$.*

One may ask if a legal state as defined above can be maintained at all times when using $\text{OPT}_{\geq 0} = \text{OPT}$. However, in this case the legal state cannot be maintained at all times. Starting with time $t = 0$, the adversary can inject B packets in nodes $[n/2, n)$ and offer edges to the nodes $[1, n/2)$ thereby moving all the optimal packets to the nodes $[1, n/2)$. Now the adversary can inject further B packets at nodes $[n/2, 3n/4)$ and for $\frac{nB}{4}$ time steps activate all edges (i, j) such that $j = i+1, n/4 \leq i \leq n-1$. This results in $f_{\geq 1}$ being undefined for the packets in the layer 1 of our algorithm and also $\text{OPT}_{\geq 1}$ might no longer be a subset of $\text{OPT}_{\geq 0}$. To rectify this situation, we view the destination node n as being a virtual set of further n nodes with a buffer size of $B+1$ at each of them. These artificially "kept alive" packets will serve to bring the system back to a legal state. We therefore define $\text{OPT}_{\geq 0}$ to be the set of all packets in the optimal algorithm plus the "kept alive" packets.

To prove that the system can always be maintained in a legal state, possibly using the artificial packets, we prove the following Lemma.

LEMMA 2.11. *If the system is in a legal state at the beginning of time step t, then it remains in a legal state at the end of time step t.*

PROOF. To prove the Lemma we consider all the possible actions that can happen during time step t. During time step t, the adversary can inject new packet(s) or activate edges. The edges activated are classified by using the following definition.

DEFINITION 2.12. *A node u is said to belong to layer i if node u has packets in layer i and no packets in layer greater than i.*

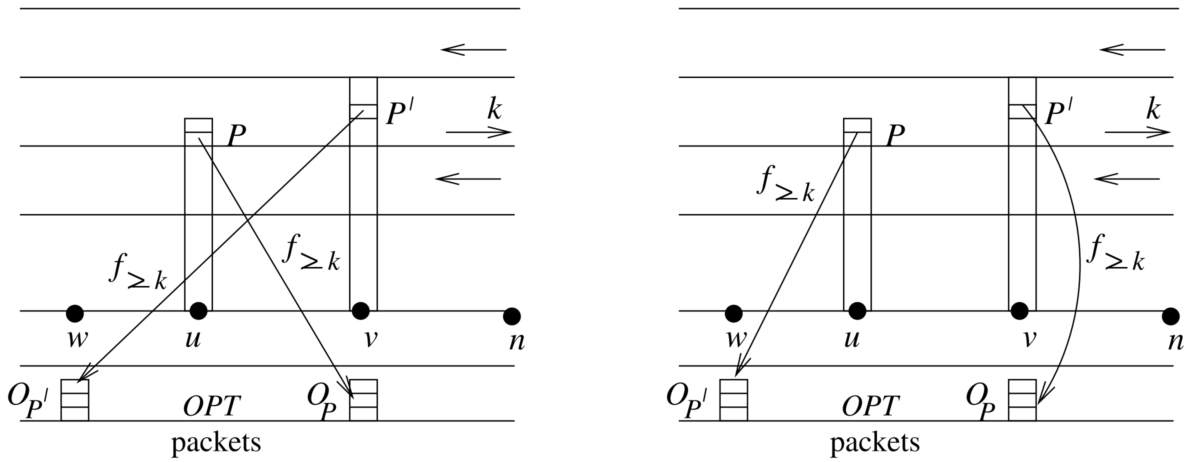
An edge $u \to v$ is called a *crossing edge* if u and v belong to different layers, otherwise it is called a *non-crossing edge*. An edge $u \to v$ is called a *schedule edge* if it is in the schedule of some packet P in the optimal algorithm, otherwise it is called a *non-schedule edge*.

Thus, edges activated by the adversary can be classified depending on whether the edge is a crossing/non-crossing edge, schedule/non-schedule edge and whether the direction of edge is in the direction or opposite to the direction of the layer to which u belongs. In each of these 8 cases we show that the legal state can always be maintained after time step t. For the purpose of the proof, we look at the activated edges during time step t one at a time and let the active edge be $u \to v$.

Whenever the system fails to be in a legal state, we perform local modifications to restore the legal state. These modifications, called the *Virtual Movement Strategy* and *Element Replacement Strategy*, are explained below.

Virtual Movement Strategy

This strategy acts as a subroutine in our proof. A call of the form $\text{VM}(v, P, k)$ means that for packet P at a node u the mapping $f_{\geq k}$ is not monotonic and hence needs to be corrected by using packets in layer k of node v. The caller also ensures that v has $B+1$ packets in layer k. This situation as shown in Figure 1(a) is a violation of the legal state as it does not represent a monotonic $f_{\geq k}$ mapping. However, we can bring the system back to a legal state as follows.



(a) (b)

Figure 1: Figure (a) shows a system not in a legal state and (b) shows the correction performed using *Virtual Movement Strategy* to bring the system back to a legal state.

Select a packet P' with $\mathrm{POS}(P') = v$ and $\mathrm{POS}(f_{\geq k}(P')) \neq v$, i.e. the position is only to the left(*right*) of v if the direction of layer k is a Always-Go-Right (Always-Go-Left). The caller ensures that such a packet P' exists at node v in layer k. Now we swap the mappings for packets P and P' as shown in Figure 1(b) thus bringing the system back to a legal state.

The strategy is called Virtual Movement Strategy because one can think of renaming P and P' after the swapping thereby moving P' virtually. So P and P' exchange their roles. We note that that Virtual Movement Strategy does not change any $\mathrm{OPT}_{\geq j}$. It is only the mapping in $f_{\geq k}$ that was changed to restore monotonicity.

Element Replacement Strategy

Suppose that during some time step, the algorithm moves a packet P from a layer i to a layer $j \leq i-1$. Then we release the $f_{\geq k}$ mappings of P for $i \leq k < j$. Suppose for some $k, i \leq k < j, f_{\geq k}(P) = O_k \neq f_{\geq k-1}(P) = O_{k-1}$ and for some packet P', $f_{\geq k}(P') = O_{k-1}$ before the movement of P by algorithm A. After the movement of P, $\mathrm{OPT}_{\geq k-1} \not\subset \mathrm{OPT}_{\geq k}$ because $O_{k-1} \in \mathrm{OPT}_{\geq k}$ and $O_{k-1} \notin \mathrm{OPT}_{\geq k-1}$. This violates the subset rule of the legal state. This situation is shown in Figure 2(a) where the label on the arc indicates the layer at which the mapping is made. To bring the system back to a legal state, we simply set $f_{\geq k}(P') = O_k$ as shown in Figure 2(b). With this adjustment $\mathrm{OPT}_{\geq k}$ now becomes $\mathrm{OPT}_{\geq k} - \{O_{k-1}\}$ and $\mathrm{OPT}_{\geq k-1}$ becomes $\mathrm{OPT}_{\geq k-1} - \{O_{k-1}\}$. Thus after the time step t also, it holds that $\mathrm{OPT}_{\geq k} \subseteq \mathrm{OPT}_{\geq k-1}$.

The strategy is called Element Replacement Strategy because we ensure that $\mathrm{OPT}_{\geq k}$ and $\mathrm{OPT}_{\geq k-1}$ lose the same element, O_{k-1}.

Below, we look at each of the 8 different categories of active edges and use the Virtual Movement and Element Replacement strategies to bring the system back to a legal state. First we look at edges that are not leading to the destination. The actions of the algorithm on edges from node $n-1$ to node n are treated separately.

1. Non-crossing Edges

a. Schedule Edges

i. In the direction of the layer:

In this case, the algorithm moves a packet P_a from u to v while the optimal algorithm moves a packet P_o from u to v. Let u and v belong to layer i. If for P_a it happens that $f_{\geq k}$ with $k < i$ and layer k opposite in the direction of layer i is no longer monotonic, this can be restored by calling $\mathrm{VM}(u, P_a, k)$. If there exists a packet P in the algorithm with $\mathrm{POS}(P) = u$ and $f_{\geq k}(P) = P_o$ at a layer k that is in the same direction as that of i, then $f_{\geq k}$ is no longer monotonic. If $k < i$, calling $\mathrm{VM}(v, P, k)$ would again make $f_{\geq k}$ monotonic. If $k = i$, we can swap $f_{\geq i}(P)$ with $f_{\geq i}(P_a)$ to restore monotonicity of $f_{\geq k}$. Since there are no changes to $\mathrm{OPT}_{\geq j}$ for any $j \geq 0$, the subset property holds after time step t.

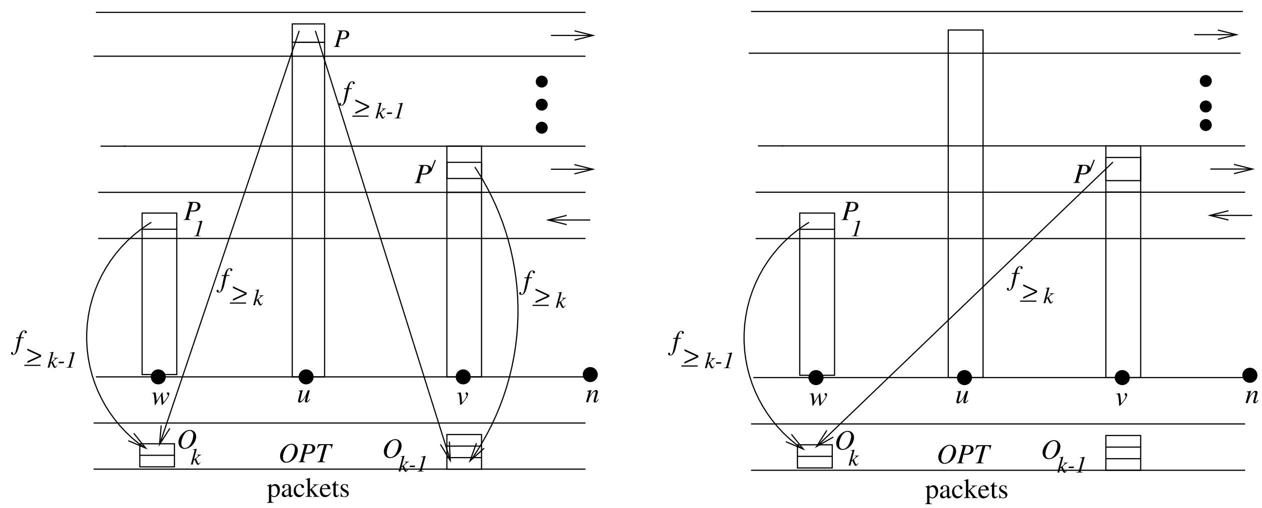
ii. Opposite to the direction of the layer:

In this case, the algorithm cannot move any packet from u to v. Let u, v belong to layer i. Let the optimal algorithm move packet P_o. If there exists a packet P in the algorithm with $\mathrm{POS}(P) = u$, $f_{\geq k}(P) = P_o$ where the direction of layer k is opposite to the direction of layer i, then $f_{\geq k}$ is no longer monotonic. This is corrected by using $\mathrm{VM}(v, P, k)$. Further, since at any layer $j \geq 0$, $\mathrm{OPT}_{\geq j}$ did not undergo any change, the subset property holds after time step t.

b. Non-schedule Edges

i. In the direction of the layer:

In this case, the optimal algorithm does not perform any packet movements. Let the algorithm move packet P_a from u in layer i. In any layer k that is in the same direction as that of i, the $\mathrm{OPT}_{\geq k}$ mapping is still valid. In layers $k, k < i$ that are in the opposite direction of layer i, $f_{\geq k}(P_a)$ might not be monotonic. Here we call $\mathrm{VM}(u, P_a, k)$ to regain monotonicity. Observe that in layer k node u has $B+1$ packets and



(a) (b)

Figure 2: The figure on the left shows the existing mapping and the sets $\mathrm{OPT}_{\geq k}$ and $\mathrm{OPT}_{\geq k-1}$ might lose different elements thereby violating the subset property. The figure on the right shows the new mappings where the sets $\mathrm{OPT}_{\geq k}$ and $\mathrm{OPT}_{\geq k-1}$ lose the same element O_{k-1}. The packets in the optimal algorithm are shown below the line graph for convenience.

hence the call to VM is valid. There would be no changes in $\mathrm{OPT}_{\geq j}$ for any $i \geq 0$ and hence the subset property holds after time step t. If the algorithm does not move any packet then, the state of the system will also be legal after time step t.

ii. Opposite to the direction of the layer:

In this case, neither algorithm A or the optimal algorithm perform any movement of packets and hence the state of the system remains legal after time step t.

2. Crossing Edges

a. Schedule Edges

i. In the direction of the layer:

In this case, the algorithm moves a packet P_a from u to v from layer i to layer j, $j \leq i-1$. The optimal algorithm moves a packet P_o from u to v. If the packet P that is mapped to P_o is at u, i.e. $\mathrm{POS}(P) = u$, then we first swap $f_{\geq k}(P)$ with $f_{\geq k}(P_a)$. This ensures that when the algorithm moves the packet P_a and the optimal algorithm moves the packet P_o, there would be no violations to the monotonicity rule. By induction hypothesis, if $\mathrm{POS}(P) \neq u$ before time step t then the movement of P_o would not result in any violations to monotonicity. We can now release the mappings of P_a for all layers $k, j < k \leq i$. This might violate the subset rule. To restore this, we use the Element Replacement Strategy. Using the two strategies as needed, we bring the system back to a legal state after time step t.

ii. Opposite to the direction of the layer:

In this case also, both the algorithm and the optimal algorithm move a packet from u to v. Let the algorithm move packet P_a from layer i to layer $j, j \leq i-1$. To correct any violations to monotonicity, we use a similar approach as in case ($2a.i$) above. We now release the mappings of P_a for layers $k, j < k \leq i$. If for any such layer k the subset rule is violated as $\mathrm{OPT}_{\geq k}$ and $\mathrm{OPT}_{\geq k-1}$ might lose a different element, we use the Element Replacement Strategy to ensure that after time step t also $\mathrm{OPT}_{\geq k} \subseteq \mathrm{OPT}_{\geq k-1}$. For any layer $k \leq j$, $\mathrm{OPT}_{\geq k}$ has no change. Thus, after the above modifications, the system remains in a legal state after time step t.

b. Non-Schedule Edges

i. In the direction of the layer:

In this case, the optimal algorithm does not move any packet but the algorithm moves a packet P_a from layer i to layer j. If the monotonicity rule is violated for P_a at a layer k opposite to the direction of layer i, we call $\mathrm{VM}(u, P_a, k)$ to correct the violation. Note that $k \leq j$ and hence the call to VM would succeed. Similarly, for layers $k, j < k \leq i$, the subset rule might be violated as we drop the $\mathrm{OPT}_{\geq k}$ mapping for the packet P_a. This situation is restored using the Element Replacement Strategy. For any layer $k \leq j$ $\mathrm{OPT}_{\geq k}$ does not change. Thus, after the above modifications the system can be brought back to a legal state after time step t.

ii. Opposite to the direction of the layer:

Let the node u belong to layer i and v belong to layer j. In this case, the algorithm moves a packet P_a whereas the optimal algorithm does not move any packet. The only violations to monotonicity can occur in layers k, with $k \leq j$ being in the same direction as layer i. This can be fixed by using $\mathrm{VM}(u, P_a, k)$. Since in layer j, u has already $B+1$ packets, the call to VM succeeds. At this point, we can release the mappings of P_a in all layers $k, j < k \leq i$. This might result in violations to the subset rule which can be corrected using the Element Replacement Strategy.

Thus in all the cases, the system remains in a legal state after time step t. The above cases, however, do not follow immediately when the movement is to the destination node. These are considered below. The movements to the destination can occur because of a schedule edge or a non-schedule edge. In the following we let d be the destination node and u be the node before the destination node.

1. Schedule edge movements to the destination: In this case, if $h(u) > 0$ then, the algorithm moves a packet P_a. The optimal algorithm moves a packet P_o. We do not have to maintain the mappings for P_a any longer. Also, if P_a is mapped to an artificial packet then we can release that mapping. Because P_o is moved, for packets P in the algorithm with $f_{\geq k}(P) = P_o$ for some Always-Go-Left layer k to restore monotonicity, we proceed as follows. If $k = 1$, then we keep the packet P_o artificially alive at the destination. If $k > 1$ then we can replace $f_{\geq k}(P)$ with $f_{\geq 1}(P)$. At this point we can release all the mappings of P_a. and any violations of the subset rule can be again corrected using the Element Replacement Strategy.

   Now consider the situation that $h(u) = 0$. In this case, as only the optimal algorithm moves a packet P_o, it might happen that for some packet P in an Always-Go-Left layer k, $f_{\geq k}(P) = P_o$. We can restore the mappings as done earlier by using the mapping at layer 1 and keeping P_o artificially alive if $k = 1$.

2. Non-schedule movements to the destination: In this case the algorithm moves a packet P_a from u to d if $h(u) > 0$. At this point we can release the mappings of P_a and correct any violations of the subset rule using the Element Replacement Strategy. The optimal algorithm does not move any packet from u in this case.

Apart from activating edges, during time step t the adversary might inject new packets. For every new packet injected at a node u, we simply store the incoming packet P in the lowest available empty slot. If node u belongs to layer i, we set $f_{\geq k}(P) = P$ for all layers $k \leq i$. □

Using lemma 2.11, we prove the following lemma.

LEMMA 2.13. *The system remains in a legal state for all time steps $t \geq 0$.*

PROOF. At $t = 0$, the state of the system is trivially legal and we can use Lemma 2.11 inductively. □

From the definition of the legal state, the Corollary below holds.

COROLLARY 2.14. *For any $i \geq 0$, $|ALG_{\geq i}| \leq |OPT_{\geq i}|$.*

PROOF. Since the system stays in a legal state at all times, it implies that the mapping from $\text{ALG}_{\geq i}$ to $\text{OPT}_{\geq i}$ is injective. Hence, it must hold that $|\text{ALG}_{\geq i}| \leq |\text{OPT}_{\geq i}|$. □

Let w_i be the number of nodes that have packets in all the $B+1$ slots of layer i. Let $l_i = |\text{POS}(\text{ALG}_{\geq i})|$. The following Lemma derives an upper bound on w_i.

LEMMA 2.15. $w_{i+1} \leq \frac{w_{i-2}}{2}$.

PROOF. Consider layer $i+1$. By definition, w_{i+1} nodes have $B+1$ packets each in layer $i+1$. Also, these w_{i+1} nodes have $B+1$ packets in layer i. Hence $\text{ALG}_{\geq i}$ has at least $2w_{i+1} \cdot (B+1)$ packets. Hence,

$$2w_{i+1} \cdot (B+1) \leq |\text{ALG}_{\geq i}| \leq |\text{OPT}_{\geq i}| \tag{1}$$

The second inequality follows from Corollary 2.14.

Since the system is in a legal state, because of the monotonicity and the subset properties it holds that the optimal packets in $\text{OPT}_{\geq i}$ can have positions only in the nodes where $\text{ALG}_{\geq i-1}$ have positions at. See also Figure 3. Thus, $\text{POS}(\text{OPT}_{\geq i}) \subseteq \text{POS}(\text{ALG}_{\geq i-1})$. Also the optimal algorithm has a buffer size of only B at each node. Hence, for $i > 0$

$$|\text{OPT}_{\geq i}| \leq l_{i-1} \cdot B \tag{2}$$

Equation (3) below follows from the definition of l_i and w_i.

$$l_i \leq w_{i-1} \tag{3}$$

Combining Equations (1), (2) and (3), the Lemma follows. □

Using Lemma 2.15, the maximum number of layers that algorithm A uses can be bounded as in the Theorem below.

THEOREM 2.16. *The maximum number of layers Algorithm A needs to maintain is at most* $3 \cdot (\log n + 1)$.

PROOF. For any $i > 1$, layer $i+1$ comes into existence only after layer i has $w_i \geq 1$. Since $w_i \leq w_{i-3}/2$, and $w_0 \leq 2n$, we have that $w_{3(\log n+1)} \leq 1$ and that bounds the maximum number of layers that Algorithm A needs. □

Theorem 2.16 gives an upper bound on the height of any node in the algorithm. It follows from the proof of Lemma 2.13 that throughput of the algorithm A is greater than or equal to that of the optimal algorithm.

Thus we arrive at the following Theorem.

THEOREM 2.17. *Algorithm A is $O(\log n)$–competitive.*

3. CYCLE GRAPH

For the cycle graph, let the nodes be numbered consecutively from 1 to n with node n being the destination node. As for the case of line graph, we show that any deterministic online algorithm for gathering in the cycle graph has a lower bound of $\Omega(\log n \cdot B)$ on the buffer size.

LEMMA 3.1. *Any deterministic online routing algorithm is $\Omega(\log n)$ competitive on a cycle graph of n nodes.*

PROOF. The proof uses similar techniques as in Theorem 2.1 and is omitted here. □

If one considers randomized online algorithms also, then the adversary creates with a probability of success at least $\frac{1}{n}$ a node of height $\Omega(\log n \cdot B)$.

3.1 Algorithm

In the algorithm, nodes have buffers that can store packets and each buffer has slots numbered consecutively starting from 1. A layer is a consecutive set of $B+1$ slots where each slot can store one packet. As in the case of the line graph, the algorithm maintains layers that alternate direction between clockwise and counter-clockwise as shown in Figure 4, starting from a clockwise layer.

The definition of the height of a packet P and the height of a node is the same as defined earlier. We also define $\text{ALG}_{\geq i}$ and $\text{POS}_{\text{ALG}}()$ and $\text{POS}_{\text{OPT}}()$ in a similar way. Since in a cycle graph there is no consistent notion of left or right, we define a distance measure as follows.

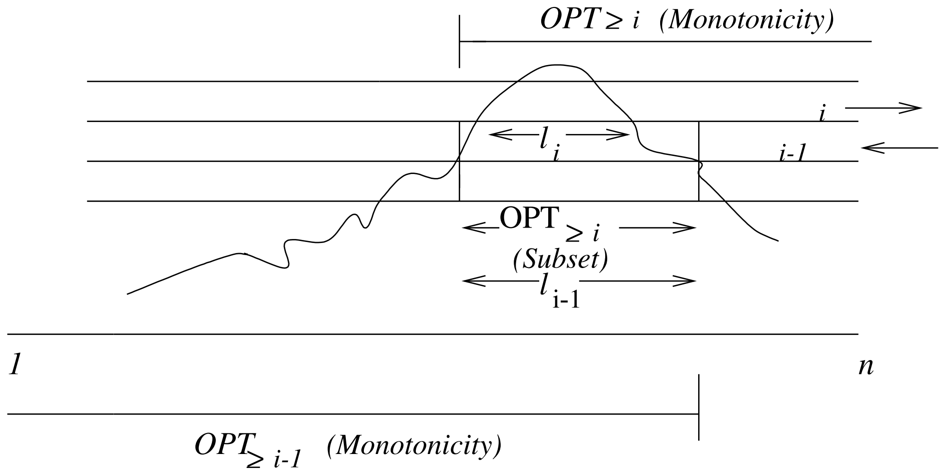


Figure 3: Showing the area of nodes for OPT$_{\geq i}$ as implied by the legal state. Using just the monotonicity rule of the legal state results OPT$_{\geq i}$ and OPT$_{\geq i-1}$ possibly having the area as marked monotonic within braces.

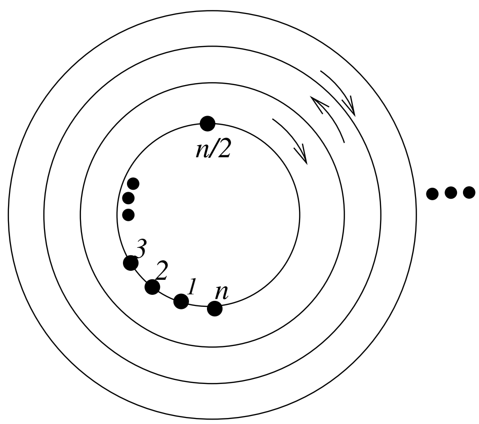


Figure 4: Figure showing the alternating clockwise and counter-clockwise layers.

DEFINITION 3.2 (DISTANCE). *For a node u, $dist_{cw}(u)$ is the length of the path from u to n in the clockwise direction. Similarly, $dist_{ccw}(u)$ is the length of the path from u to n in the counter-clockwise direction.*

Similar to left(*right*) monotonic mapping we define clockwise (*counter-clockwise*) monotonic mapping as follows.

DEFINITION 3.3 (MONOTONIC MAPPING). *Given two packet sets $\mathcal{P}, \mathcal{P}'$, a mapping $f : \mathcal{P} \to \mathcal{P}'$ is called clockwise(resp. counter-clockwise)-monotonic if for all $P \in \mathcal{P}$, $dist_{cw}(P) \leq dist_{cw}(f(P))$. (resp. $dist_{ccw}(P) \leq dist_{ccw}(f(P))$).*

We define $f_{\geq i}$: ALG$_{\geq i}$ $\to$ OPT$_{\geq i}$ as a monotonic mapping in the direction of layer i. Now we present Algorithm B for performing gathering in a cycle graph.

Algorithm B
1. For every time step and every edge (u, v) from u to v
2. if there exists a layer l with the direction same as the direction of (u, v) and u has at least 1 packet in l and v has at most B packets in l
3. move a packet from u to v
4. For every time step at every node u
5. Accept incoming and injected packets and store them at lowest available empty slots currently.

End Algorithm

Similar to the case of line graph, algorithm B also can move not only the highest available packet but also a packet from any height. It can be observed that allowing only the highest available packet to move may result in the system having unbounded number of packets. Fact 2.8 also holds in the case of Algorithm B. In the next subsection we show that algorithm B has an upper bound and also a lower bound of $O(\log n)$ on the competitive ratio.

3.2 Proof

The proof needs some extra tricks. In contrast to Lemma 2.9 the number of packets in the algorithm is not bounded by the number of packets in the optimal algorithm.

We first show an upper bound on the number of packets that can be in the system at any time. Then, we define a legal state for the system and show that the system will stay in a legal state at any time. We use the legal state to show that algorithm B has a competitive ratio of $O(\log n)$.

LEMMA 3.4. *The total number of packets in the system at any time is at most $2n \cdot (B + 1)$.*

PROOF. Let OPT denote the set of packets in the optimal algorithm. The set OPT can be divided into two disjoint subsets OPT$_{cw}$ and OPT$_{ccw}$ where OPT$_{cw}$(*resp.* *OPT*$_{ccw}$) contains the set of optimal packets that are delivered to the destination through the clockwise direction (*resp.* counter-clockwise direction). Note that OPT = OPT$_{cw}$ $\cup$ OPT$_{ccw}$. Similarly, the set of packets in the algorithm, ALG, can also be divided into two disjoint subsets A_{cw} and A_{ccw}. We provide two mappings $f_{cw} : A_{cw} \to$ OPT$_{cw}$ and $f_{ccw} : A_{ccw} \to$ OPT$_{ccw}$ with the following properties.

- Both f_{cw} and f_{ccw} are injective and monotonic in clockwise and counter-clockwise directions respectively.
- $\forall P \in A_{ccw}, h(p) > (B + 1)$

- $\forall P \in$ ALG, if $h(p) > (B+1)$ then $P \in A_{cw}$ or $P \in A_{ccw}$.

If f_{cw} and f_{ccw} satisfy the above properties then they are also called *valid*. Since it is not possible to provide the mapping for all packets in ALG$_{\geq 0}$ we classify packets in ALG$_{\geq 0}$ as leaders and losers as defined below. We say that a packet P in the algorithm is a *leader* if it belongs to either A_{cw} or A_{ccw} and a *loser* otherwise. The validity of the mappings also imply that for any packet P that is a loser, $h(P) \leq B+1$.

Claim 3.5 shows that the mappings defined above can be maintained validly at all times. Since loser packets are restricted to only the first layer in the algorithm and the mappings are injective,

$$\begin{aligned} |\text{ALG}| &\leq |A_{cw}| + |A_{ccw}| + n \cdot (B+1) \\ &\leq |\text{OPT}_{cw}| + |\text{OPT}_{ccw}| + n \cdot (B+1) \\ &\leq |\text{OPT}| + n \cdot (B+1) \\ &\leq 2n \cdot (B+1) \end{aligned}$$

Hence the proof. □

CLAIM 3.5. *For all time steps t and over any sequence of packet injections and edge activations σ, the mappings f_{cw} and f_{ccw} can be maintained validly.*

PROOF SKETCH. To simplify the presentation, we can think of all packets in A_{cw} and OPT$_{cw}$ that are mapped under f_{cw} are colored blue and packets in A_{ccw} and OPT$_{ccw}$ are colored green. During all time steps, we maintain the mappings valid by showing that blue packets get mapped to blue packets and green packets get mapped to green packets. The proof is thus done by induction on time steps t. At $t = 0$, the mappings are trivially valid. Assume that mappings are valid till time $t-1$. At time t, we compare all the actions of the algorithm with those of the optimal algorithm. Any violations to the validity of the mappings are then restored using local modifications.

Consider edge activations during time step t. All the edges that are activated by the adversary can be classified into 8 categories as in the proof of Lemma 2.11. For each of the cases, we show that the mappings remain valid. In the following, we look at active edges one at a time and $u \rightarrow v$ represents the active edge. In this sketch, we look at just one case. The treatment for the other cases is similar and omitted here.

Non-crossing, schedule edge in the direction of the layer:

We have several cases to consider. Let u and v belong to layer 0. In this case, let the optimal algorithm move packet P_0 and algorithm B move packet P_a. Since we need not provide f_{ccw} mapping for packets in layer 0, we need to look at violations to validity of f_{cw}. f_{cw} will not be valid if there is a packet P in the algorithm with POS$(P) = u, f_{cw}(P) = P_o$. This can be easily corrected by swapping $f_{cw}(P)$ with $f_{cw}(P_a)$. In the case that the algorithm did not move any packet, this implies that $h(v) = B+1$ and hence we can always find a packet P' with POS$(P') = v$, POS$(f_{cw}(P')) \neq v$ and swap $f_{cw}(P)$ with $f_{cw}(P')$.

If u and v belong to a layer greater than 0, let the algorithm move packet P_a and the optimal algorithm move packet P_o. If P_a and P_o are of the same color, then we can make any required corrections as in the above. On the other hand, if P_a and P_o are of different color, we proceed as follows. W.l.o.g, let P_a be a green packet and P_o be a blue packet and the layer that u and v belong is in the clockwise direction. If in the algorithm there are no blue packets at u, then from all the green packets at u there must be at least one green packet P' with POS$(f_{ccw}(P')) \neq u$. Hence we can swap $f_{ccw}(P)$ with $f_{ccw}(P')$ to make the mapping valid. If there is at least one blue packet P_1 at u, then we swap the colors of P_1 and P_a and then swap $f_{cw}(P_1)$ and $f_{ccw}(P_a)$. Also, the movement of P_o might violate the validity of the f_{cw} mapping and in this case we look at if there are any blue packets in the algorithm at node v or not and in each case correct the violation.

In all the other cases also, we perform similar corrections. The modification used are to find a packet and swap the mappings or if this is not possible, find a differently colored packet and swap colors and the mappings. Also, for movements to destination since the mappings are monotonic whenever an optimal packet reaches the destination a packet in the algorithm would also reach the destination at that times step or earlier.

Apart from edge activations, we need to consider packet injections. For a packet injected at node u during time step t, we set $f_{cw}(P) = P$ and if $h(u) > B+1$, we also set $f_{ccw}(P) = P$. Thus the mappings can be maintained validly after time step t also.

The similarity with the proof of Lemma 2.9 can be observed in the sense that in a line graph there is only one path to destination for any packet and hence maintaining one monotonic mapping $f_{\geq 0}$ suffices to obtain an upper bound on the number of packets in the system. But for the case of the cycle graph, there are two possible paths to destination(through the clockwise and counter-clockwise directions) and hence we needed to maintain a monotonic mapping in both the directions. □

We now define a legal state for the system.

DEFINITION 3.6 (LEGAL STATE). *A system is said to be in a legal state if it satisfies:*

1. *Monotonicity: For all $i \geq 0$, $f_{\geq i}$ is injective and monotonic, and*
2. *Subset: For all $i \geq 0$, $OPT_{\geq i+1} \subseteq OPT_{\geq i}$.*

But it can be observed that the legal state as defined above cannot be maintained at all times. Since there are two paths to the destination from any node u, the adversary can inject packets at a node and while the optimal algorithm delivers them to the destination in the direction opposite to that of the current layer. This violates the injective property and monotonicity of the mapping as well as the subset rule. To correct this situation, similar to the case of line graph, we view the destination node n as a set of $2n$ virtual nodes each having a buffer of size $B+1$. For these virtual nodes, we define dist$_{cw}$ as well as dist$_{ccw}$ to be n. Packets reaching these virtual nodes are kept alive artificially. And we define OPT$_{\geq 0}$, OPT$_{\geq 1}$ as set of all optimal packets in the system and the kept alive packets. Thus we have that $|\text{OPT}| \leq n \cdot B$ and $|\text{OPT}_{\geq 0}| \leq 3n \cdot (B+1)$. These artificially "kept alive" will serve to bring the system back to a legal state.

To prove that the system can always be maintained in a legal state, we prove the following lemma.

LEMMA 3.7. *For any sequence of packet injections and edge activations, Algorithm B guarantees that the system in a legal state for all time steps.*

PROOF SKETCH. We start with classifying the active edges depending on whether the edge is crossing/non-crossing and whether the edge is schedule/non-schedule. This classification results in 8 classes of edges as in proof of Lemma 2.11. For each of the above cases, using strategies similar to Virtual Movement Strategy and Element Replacement Strategy we show that if the state of the system is legal before a time step t, then the state of the system is legal after time step t also. For packets injected during any time step t

we can easily provide the monotonic mappings that satisfy the requirements of the legal state. Observing that at $t = 0$, the system is in a legal state trivially, a simple induction on time would then prove the lemma. $\square$

From the above discussion and techniques similar to proof of Lemma 2.15, the following theorem follows.

THEOREM 3.8. *Algorithm B achieves a competitive ratio of $4 \log n$.*

4. CONCLUSION

We demonstrated in this paper that highly competitive routing algorithms can be designed for adversarial systems based on lines and cycles. It would be interesting to investigate also other topologies, such as trees, meshes, etc. Also, it would be interesting to study how to handle a moving observer or several observers. We expect the sensor network area to be full of many exciting future problems in the area of routing theory.

5. REFERENCES

[1] Factoid project. Available at http://www.research.digital.com/wrl/projects/Factoid.

[2] PicoRadio project. Available at http://bwrc.eecs.berkeley.edu/Research/Pico_Radio/Default.htm.

[3] Ultra Low Power Wireless Sensor project. Available at http://www-mtl.mit.edu/∼jimg/project_top.html.

[4] WINS project. Available at http://www.janet.ucla.edu/WINS.

[5] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Adaptive packet routing for bursty adversarial traffic. In *Proc. of the 30th ACM Symp. on Theory of Computing (STOC)*, pages 359–368, 1998.

[6] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proc. of the 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 380–389, 1996.

[7] E. Anshelevich, D. Kempe, and J. Kleinberg. Stability of load balancing algorithms in dynamic adversarial systems. In *Proc. of the 34th ACM Symp. on Theory of Computing (STOC)*, 2002.

[8] B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple online strategies for adversarial systems. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, 2001.

[9] B. Awerbuch and F. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. of the 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 459–468, 1993.

[10] B. Awerbuch and F. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proc. of the 26th ACM Symp. on Theory of Computing (STOC)*, pages 487–496, 1994.

[11] B. Awerbuch, Y. Mansour, and N. Shavit. End-to-end communication with polynomial overhead. In *Proc. of the 30th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 358–363, 1989.

[12] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proc. of the 28th ACM Symp. on Theory of Computing (STOC)*, pages 376–385, 1996.

[13] D. Gamarnik. Stability of adversarial queues via fluid models. In *Proc. of the 29th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 60–70, 1998.

[14] D. Gamarnik. Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 206–214, 1999.

[15] A. Goel. Stability of networks and protocols in the adversarial queueing model for packet routing. In *Proc. of the 10th ACM/SIAM Symp. on Discrete Algorithms (SODA)*, pages 911–912, 1999.

[16] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proc. of Hawaiian International Conference on Systems Science*, 2000.

[17] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of the 6th ACM/IEEE Mobicom Conference*, pages 56–67, 2000.

[18] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Maximum lifetime data gathering and aggregation in wireless sensor networks. Technical report TR CS-02-12, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, August 2002.

[19] R. Katz, J. Kahn, and K. Pister. Emerging challenges: Mobile networking for “Smart Dust”. *Journal of Communications and Networks*, September 2000.

[20] J. Kulik, W. Rabiner, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. of the 5th ACM/IEEE Mobicom Conference*, pages 174–185, 1999.

[21] S. Lindsey and C. Raghavendra. PEGASIS: Power Efficient GAthering in Sensor Information Systems. In *Proc. of IEEE Aerospace Conference*, 2002.

[22] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[23] C. Scheideler and B. Vöcking. From static to dynamic routing: efficient transformations of store-and-forward protocols. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 215–224, 1999.

[24] S. Singh, M. Woo, and C. Raghavendra. Power-aware routing in mobile ad-hoc networks. In *MOBICOM ’98*, 1998.

[25] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, pages 16–27, October 2000.

[26] P. Tsaparas. Stability in adversarial queueing theory. Master’s thesis, Dept. of Computer Science, University of Toronto, 1997.