Greedy Packet Scheduling

Israel Cidon* Shay Kutten† Yishay Mansour‡ David Peleg§

Abstract

Scheduling packets to be forwarded over a link is an important subtask of the routing process both in parallel computing and in communication networks. This paper investigates the simple class of *greedy* scheduling algorithms, namely, algorithms that always forward a packet if they can. It is first proved that for various "natural" classes of routes, the time required to complete the transmission of a set of packets is bounded by the sum of the number of packets and the maximal route length, for any greedy algorithm (including the arbitrary scheduling policy). Next, tight time bounds of $\Theta(n)$ are proved for a specific greedy algorithm on the class of shortest paths in n-vertex networks. Finally it is shown that when the routes are arbitrary, the time achieved by various "natural" greedy algorithms can be as bad as $\Omega(n^{1.5})$, when $O(n)$ packets have to be forwarded on an n-vertex network.

*IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, and Faculty of Electrical Engineering, The Technion, Haifa 32000, Israel. cidon@techsel.bitnet

†IBM T.J. Watson Research Center P.O. Box 704, Yorktown Heights, NY 10598, kutten@ibm.com

‡Laboratory for Computer Science, MIT, Cambridge, MA 02139. Partially supported by IBM graduate fellowship. mansour@theory.lcs.mit.edu

§Department of Applied Mathematics, The Weizmann Institute, Rehovot 76100, Israel. peleg@wisdom.bitnet. Supported in part by an Allon Fellowship, by a Walter and Elise Haas Career Development Award and by a Bantrell Fellowship. Part of the work was done while visiting IBM T.J. Watson Research Center.

1 Introduction

The task of managing the delivery of packets in a distributed communication network is intricate and complex. Consequently, many routing strategies incorporate design choices directed at simplifying the process. One prime example for this type of choice is the decision to create a clear distinction between two subtasks, namely, *route selection* and *packet scheduling*. The first subtask involves selecting for each packet the route it should use from its source to its destination. This selection is done in advance, before the packet actually leaves its source. The second subtask concerns the transmission stage itself, and involves deciding on the schedule by which the different packets are to be forwarded over each edge along their routes. At this stage, the packets are restricted to their predetermined routes, and cannot deviate from them. This paper concentrates on routing strategies adopting this separation, henceforth referred to as *fixed-route strategies*, and in particular on the scheduling subtask.

A second type of design choice, aimed at simplifying the scheduling process considerably, is to make scheduling decisions *locally* and per packet, rather than globally. The scheduling policy is thus restricted to the selection of local rules for managing the queues on outgoing links, namely, resolving the conflicts between the different packets that need to be advanced on the same outgoing edge. Intuitively, a *local* algorithm has the property that the rules used by a node in order to schedule awaiting packets rely only on information concerning these packets (typically contained in the packets' headers), such as the identity of the source and destination, the distance traversed by the packet so far, the arrival time at the current node etc. In contrast, a global algorithm can base its decisions on additional global information on the status of the network, such as the current distribution of packets in the network and the routes of these packets.

Although the two design decisions discussed above may not generally lead to a globally optimal algorithm, they are both widely used. In fact, one of the main distributed network strategies for packet routing is *virtual circuit switching* [CGK88, Mar82, BG87], which is based on fixing a single predetermined *logical circuit* from the given source to the given destination, and transmitting *all* packets between them on this circuit. While setting up the circuit, the sender and the destination do not know which other logical circuits will overlap their own circuit in the duration of its existence. Nontheless, similar considerations apply also for the second common routing strategy, known as *packet switching* [MRR80]. In a fixed-route packet switching strategy, different packets going from the same source to the same destination may traverse different paths. Thus in such strategy, congestion over a link does influence the selection of routes for later packets, although it cannot change the routes of packets that are already in transit.

Fixed-route strategies are employed in communication networks such as SNA [Mar82], APPN [BGGJP85] and TYMNET [BG87]. As for the scheduling policy, most networks use a combination

of FIFO, certain priority parameters, and flow control information, to determine the next packet to be forwarded. All of these mechanisms are "approximately" local greedy (although flow control adds some global flavor). The main reasons for these choices are based on their advantages from an engineering point of view, namely, their simplicity and low complexity (compared to the global approach). The costs of the alternative, global approach are high due to the need to synchronize the network (so that schedules for entire routes be meaningful), the need to accumulate information about packets that are to be sent (and the associated delay until the information is accumulated), and related factors.

Packet scheduling algorithms for fixed-route strategies were studied by Leighton, Maggs and Rao [LMR88]. Although motivated by routing problems in specific networks realizing parallel machines, the paper studies the problem on networks of arbitrary topology. The results of [LMR88] demonstrate the fact that fixed-route strategies have advantages not only from an engineering point of view, but from a theoretical point of view as well. The problem is formalized in [LMR88] as follows. Initially, there are k packets in the network, each is assigned a path of length d, and no edge in the network is to be traversed by more than c packets. The scheduling algorithm has to forward the packets along the assigned paths.

The first result of [LMR88] is a proof that there exists a schedule that terminates in $O(d+c)$ time. However, it seems that determining this schedule requires a complex *centralized* computation, relying on global information. The paper provides also some *randomized* distributed protocols for the problem. These protocols are simple, online and local (in the sense discussed above). The first applies to arbitrary sets of paths and requires $O(c + d\log|V|)$ time (where $|V|$ is the number of nodes), and the second completes the routing in $O(c+l+\log|V|)$ time, and applies to the case when the paths are leveled with l levels. (Informally, a set of paths is *leveled* if the nodes of the network can be partitioned into levels in such a way that each edge of the paths connects two consecutive levels.)

In contrast with both types of algorithms considered in [LMR], in this paper we consider the complexity of *deterministic distributed* algorithms. In fact, we concentrate on a class of very simple on-line routing algorithms, termed *greedy* algorithms. These are algorithms that will always forward some message over each link whenever they can (i.e., whenever there is a message waiting to be forwarded on that link). The class of greedy policies is very natural [Ko78], and in fact, all routing policies used in practical systems of which we are aware fall in this class. Greedy algorithms differ in the rule employed locally for deciding on the packet to be forwarded, in the case that more than one packet awaits to be sent over the same link.

In the sequel we present several results concerning the behavior of greedy scheduling algorithms. We first look at some restricted path classes. To begin with, in Section 3 we show that for a *leveled*

set of paths, the time required for routing k packets on paths of length d by any greedy algorithm is at most $d+k-1$. This holds even for an arbitrary (yet greedy) scheduling policy, or put another way, even when an adversary is permitted to select the next packet to be forwarded in each step. We actually establish an appropriate extension of this result, applying to the more general case where packets start at different times and traverse routes of different lengths. The time by which the i'th packet arrives its destination is bounded by $\tau_i^A + d_i + k - 1$, where d_i is the length of the ith packet's route and τ_i^A is the start time of the i'th packet. This implies the same result for the natural case of *unique subpaths*. (A collection of routes enjoys the *unique subpaths* property if for every pair of nodes u and v, whenever two different routes both have a segment connecting u and v then these two segments are identical.)

In Section 4 we consider the class of *shortest paths*, where each route is required to be a *shortest path* between its source and destination. For this class, we present a strategy that requires at most $d+k-1$ time. Again, an appropriate extension holds for the case of varying route lengths (specifically, the arrival of the i'th packet at its destination occurs within $d_i + k - 1$ time units). This strategy is based on advancing the packet that has progressed the least so far. We conjecture that the same bound is true for any greedy algorithm.

We then turn to general route classes, where the set of paths is not restricted in any way. In contrast with the special cases discussed above, we show in Section 5 that greedy algorithms might behave badly for an *arbitrary* set of paths. This is true even when we consider natural greedy schedulers, like fixed priority, FIFO, or preferring the packet that traversed the minimum (or maximum) distance so far. We show that in such a case the time may be $\Omega(d\sqrt{k}+k)$. These negative results hold even for the case where both $k = \Theta(|V|)$ and $d = \Theta(|V|)$. This strengthens the counter-examples given in [LMR88] for the case of long routes and a large number of packets.

2 Model

We view the communication network as a directed graph, $G = (V, E)$, where an edge (u, v) represents a bidirectional link connecting the processors u and v. We assume synchronous communication, i.e., the system maintains a global clock, characterized by the property that a packet sent at time t is received by time $t+1$.

Next let us define formally the routing problem and its relevant parameters. The input to the problem is a collection $\mathcal{P}$ of k packets p_i and k associated routes ρ_i, $1 \leq i \leq k$. Packet p_i is originated at node A_i, its destination is B_i, and it is transmitted along the route ρ_i. We deal with node-simple (or, loop-free) routes. The length of the route ρ_i is d_i, and we denote $d(\mathcal{P}) = \max_i\{|d_i|\}$.

Two packets are said to *collide* at time t if they are currently waiting at the same node to be sent over the same link. The scheduling algorithm has to decide at each time t which packet to forward at time t. (Note that the paths are fixed, and hence the algorithm has no choice with respect to the edges that a packet traverses.) Let τ_i^A denote the time at which packet p_i was sent from its originator A_i, and let τ_i^B denote the arrival time of p_i at its destination B_i. Let T_i denote the time elapsing from τ_i^A until τ_i^B, i.e., $T_i = \tau_i^B - \tau_i^A$. The *schedule time* of $\mathcal{P}$ is $T(\mathcal{P}) = \max_i\{T_i\}$.

Some of our results apply only to special path types. Below we characterize these route classes.

A set of paths $\mathcal{P}$ is leveled if there exists an assignment $level : V \rightarrow [1, \cdots, |V|]$, such that for each path $\rho = (v_1 \ldots v_l)$, $level(v_j) = level(v_{j-1}) + 1$. A directed graph is leveled if there exists an assignment $level$, such that for every directed edge (u, v), $level(u) + 1 = level(v)$. In a leveled directed graph, every set $\mathcal{P}$ of routes is leveled.

The path ρ_i is a *shortest path* if its length equals the distance between its endpoints A_i and B_i. A set of paths $\mathcal{P}$ is *shortest* if every path $\rho_i \in \mathcal{P}$ is a shortest path.

A set of paths $\mathcal{P}$ has the *unique subpaths* property if for every pair of nodes u and v, all the *subpaths* connecting them in any path of $\mathcal{P}$ are identical; that is, if both the routes ρ_i and ρ_j go through u and v, then the segments of the paths connecting u and v are identical.

Finally, let us formally define the concept of greedy packet scheduling algorithms. A *greedy algorithm* is an algorithm satisfying the property that at each time unit, the set of packets that are forwarded is maximal, i.e., if there are messages waiting to be forwarded on some link then one of these messages is forwarded. Note that this includes also an algorithm that selects the message to be forwarded next on each link arbitrarily from among the waiting messages (or alternatively, allows an adversary to decide which packet will be sent next). The only restriction on the adversary is that it must send some packet out of the available ones, on every link.

3 Leveled routing

In this section we prove our first result, concerning greedy scheduling on leveled paths.

Theorem 3.1 *Let $\mathcal{P}$ be a set of k leveled paths. Then for any greedy algorithm used for routing $\mathcal{P}$,*

1. *every packet p_i arrives within $T_i \leq d_i + k - 1$ time units, and*
2. *the algorithm has schedule time $T(\mathcal{P}) \leq d(\mathcal{P}) + k - 1$.*

Proof: The intuition behind the proof arises from considering, for any given time during the execution, the levels that are *occupied* at that time (i.e., that contain at least one packet). We observe that at every time unit there is some progress, in the sense that either the number of occupied levels grows, or the lowest occupied level (the one whose number is the smallest) becomes unoccupied.

Let us first define some terminology. For each packet p_i, let $level(p_i, t)$ denote the number of the level where p_i resides at time t. A level L is said to be *occupied* at time t if there exists a packet p_i such that $level(p_i, t) = L$.

We adopt the convention that at any time $t > \tau_i^B$, $level(p_i, t)$ is incremented by one. This can be thought of as if the packet continues progressing indefinitely along some path ρ_i' extended from the destination B_i and dedicated to it, and hence never collides afterwards. This does not restrict generality in any way, since such an extension ρ_i' of the packet's route has no influence on the routes of other packets, and the arrival time of the packet is still considered to be τ_i^B, the time it has reached its original destination B_i.

Consider the collection $\mathcal{L}(t)$ of occupied levels at time t. We break this collection into "blocks" of consecutive levels (separated by unoccupied levels). We define the following parameters for each packet p_i:

- $B(p_i, t)$ is the *block* of p_i at time t (i.e., the block containing $level(p_i, t)$).

Suppose that $B(p_i, t) = \{L, L+1, \ldots, H\}$. Then

- $min(p_i, t) = L - 1$ is the maximal level that is smaller than $level(p_i, t)$ and is not occupied at time t.
- $max(p_i, t) = H + 1$ is the minimal level greater than $level(p_i, t)$ that is not occupied at time t.
- $width(p_i, t) = |B(p_i, t)| = max(p_i, t) - min(p_i, t) - 1$ $(= H - L + 1)$ is the number of levels in p_i's block, $B(p_i, t)$.

The following potential function is defined per packet p_i:

$$\Phi(p_i, t) = min(p_i, t) + width(p_i, t).$$

We are interested in the changes in this potential from one step to the next. Consequently, let us denote

$$\begin{aligned}
\Delta_\Phi(p_i, t, t') &= \Phi(p_i, t') - \Phi(p_i, t), \\
\Delta_{min}(p_i, t, t') &= min(p_i, t') - min(p_i, t), \\
\Delta_{width}(p_i, t, t') &= width(p_i, t') - width(p_i, t),
\end{aligned}$$

hence

$$\Delta_\Phi(p_i, t, t') = \Delta_{min}(p_i, t, t') + \Delta_{width}(p_i, t, t'). \quad (1)$$

The proof goes along the following lines. We show that $\Delta_\Phi(p_i, t, t+1) \geq 1$, and therefore $\Delta_\Phi(p_i, \tau_i^A, \tau_i^B) \geq T_i$. We then show that $\Delta_\Phi(p_i, \tau_i^A, \tau_i^B) \leq k + d_i - 1$. Combining the two claims implies that $T_i \leq k + d_i - 1$.

Claim 3.2 $\Delta_{width}(p_i, t, t+1) \geq 0$ *for every* $t \geq 0$.

Proof: Since the algorithm is greedy, we are guaranteed that if the levels $L, L+1, \ldots, H$ are occupied at time t, then the levels $L+1, \ldots, H+1$ are occupied at time $t+1$. Also, $L \leq level(p_i, t) \leq H$ implies $L \leq level(p_i, t+1) \leq H+1$, and therefore $L+1, \ldots, H+1 \in B(p_i, t+1)$. This implies that $width(p_i, t)$ is monotonically non-decreasing with t, and the claim follows. ∎

Unfortunately, $min(p_i, t)$ may decrease with time. We now show that the sum of the two parameters, $\Phi(p_i, t)$, is monotonically increasing.

Claim 3.3 $\Delta_\Phi(p_i, t, t+1) \geq 1$ *for every* $t \geq 0$.

Proof: As time progresses from t to $t+1$, $min(p_i, t)$ can either grow by 1, remain the same or decrease. We analyze these cases one by one. First assume that $min(p_i, t+1) = min(p_i, t) + 1$. Then $\Delta_{min}(p_i, t, t+1) = 1$, and by Claim 3.2, $\Delta_{width}(p_i, t, t+1) \geq 0$, implying the claim by Eq. (1).

Next, suppose $min(p_i, t+1) = min(p_i, t)$. This implies that level $min(p_i, t) + 1$ is occupied at time $t+1$, and hence $width(p_i, t)$ grows by at least one (since $max(p_i, t)$ also becomes occupied at time $t+1$). Thus $\Phi(p_i, t+1) - \Phi(p_i, t) \geq 1$.

The remaining case is that $min(p_i, t+1) = min(p_i, t) - x$, where $x > 0$. There is only one way for that to happen, namely, there has to be a packet in level $min(p_i, t) + 1$ that did not progress, and the block $B(p_i, t)$ is "joined" by the block just below it. But in this case, the decrease of $min(p_i, t)$ by x is offset by an increase of $width(p_i, t)$ by $x+1$. This holds since there are x more levels below $min(p_i, t)$ in the united block $B(p_i, t+1)$, and $max(p_i, t)$ becomes occupied. Hence $width(p_i, t+1) - width(p_i, t) \geq x+1$. By Eq. (1), the net change in $\Phi(p_i, t)$ satisfies

$$\Delta_\Phi(p_i, t, t+1) = \Delta_{min}(p_i, t, t+1) + \Delta_{width}(p_i, t, t+1) \geq (-x) + (x+1) = 1.$$

∎

Corollary 3.4 $\Delta_\Phi(p_i, \tau_i^A, \tau_i^B) \geq T_i$.

This corollary is complemented by the following claim, bounding the increase in Φ from above.

Claim 3.5 $\Delta_\Phi(p_i, \tau_i^A, \tau_i^B) \leq d_i + k - 1$.

Proof: Consider a packet p_i whose origin A_i is at level

$$L_A = level(p_i, \tau_i^A)$$

and whose destination B_i is at level

$$L_B = level(p_i, \tau_i^B) = level(p_i, \tau_i^A) + d_i.$$

Initially, $min(p_i, \tau_i^A) \geq L_A - width(p_i, \tau_i^A)$. On the other hand, upon arrival at the destination, $min(p_i, \tau_i^B) \leq L_B - 1$. Hence

$$\Delta_{min}(p_i, \tau_i^A, \tau_i^B) \leq (L_B - 1) - (L_A - width(p_i, \tau_i^A)) = d_i + width(p_i, \tau_i^A) - 1.$$

Since $width(p_i, \tau_i^B) \leq k$,

$$\Delta_{width}(p_i, \tau_i^A, \tau_i^A + T) \leq k - width(p_i, \tau_i^A).$$

The claim now follows by Eq. (1) above. ∎

Combining Corollary 3.4 and Claim 3.5, we get

$$T_i \leq \Delta_\Phi(p_i, \tau_i^A, \tau_i^B) \leq d_i + k - 1.$$

This completes the proof of Part (1) of the Theorem. Part (2) follows immediately from Part (1). ∎

The natural class of paths with the unique subpaths property can be analyzed using the above theorem.

Corollary 3.6 *Let $\mathcal{P}$ be a set of k paths satisfying the unique subpaths property. Then for any greedy algorithm used for routing $\mathcal{P}$,*

1. *every packet p_i arrives within $T_i \leq d_i + k - 1$ time units, and*
2. *the algorithm has schedule time $T(\mathcal{P}) \leq d(\mathcal{P}) + k - 1$.*

Proof: We prove that the delay suffered by any packet p_i is no greater than in an execution on a leveled graph (with the same k and d_i). Note that the subgraph induced by the route of any particular packet p_i plus all the packets it encounters, is leveled. One obstacle is that every other packet p_j may suffer some delay as a result of collisions before its route joins that of p_i. (A similar

event can also happen after they part, but this need not interest us, since it does not affect p_i.) This problem can be mended by extending the path of any such packet p_j in the induced leveled graph, before it meets the path of p_i. Notice that we have not changed the length of the route of packet p_i. Thus, by Part (1) of Theorem 3.1 the delay suffered by p_i in the unique subpaths case is the same as the one in the leveled paths case we have constructed. ∎

4 Shortest path routing

In this section we consider a scheduling algorithm for the case in which each route ρ_i in the set $\mathcal{P}$ uses a shortest path from its origin to its destination. We shall assume that all packets start at the same time, i.e., $\tau_i^A = 0$ for $1 \leq i \leq k$. For every time $0 \leq t \leq T_i$, let $d_i(t)$ denote the distance traversed by p_i by time t (note that in particular, $d_i(T_i) = d_i$). If p_i and p_j "collide" at time t, the algorithm resolves the collision by preferring p_i iff

$$d_i(t) < d_j(t) \text{ or } (d_i(t) = d_j(t) \text{ and } i < j).$$

We refer to this algorithm as the *Min Went* algorithm. The rest of this section is devoted to proving the following theorem.

Theorem 4.1 *If the set of paths $\mathcal{P}$ consists of shortest paths and $\tau_i^A = 0$ for $1 \leq i \leq k$ (i.e., all the packets start at the same time) then the Min Went scheduling algorithm guarantees*

1. *Every packet p_i arrives at time $T_i \leq d_i + k - 1$.*
2. *the schedule time is $T(\mathcal{P}) \leq d(\mathcal{P}) + k - 1$.*

We begin the proof by pointing out the following trivial fact regarding the relationship between packets in consecutive collisions.

Fact 4.2 *If p_i and p_j collide twice (at times t_1 and t_2), then the relation between $d_i(t)$ and $d_j(t)$ remains the same.*

Proof: The claim follows from the fact that $d_i(t_2) - d_i(t_1) = d_j(t_2) - d_j(t_1)$, since otherwise, one of the routes is not a shortest path. ∎

Definition 4.3 *Given an execution of the algorithm the* collision relation *C is defined as the collection of all triples $\langle p_i, p_j, t \rangle$ such that at time t packets p_i and p_j collide (i.e., they are at the same node, waiting for the same edge), and p_i wins the collision resolution and gets to use the edge (at time t).*

Since only one packet can go on a specific edge at a time t we can deduce the following fact.

Fact 4.4 *For every p, t there is at most one triple $\langle p', p, t\rangle$ in C.*

Consider some packet p_{i_0}. If this packet is never delayed, then $T_{i_0} = d_{i_0}$ and we are done. Hence suppose the packet was delayed along its route. We now define a *delay sequence* for p_{i_0} on the run. Let t_0 be the last time that packet p_{i_0} was delayed. (Note that such a time exists since the run is finite; a bound of $T(\mathcal{P}) \leq k \cdot d(\mathcal{P})$ on the scheduling time of any algorithm is trivial.) Namely, there is a triple $\langle p_{i_1}, p_{i_0}, t_0\rangle$ in C, and there is no such triple for p_{i_0} in later times $t > t_0$. (Recall that by Fact 4.4 there is only one such p_{i_1}.) Let t_1 be the last time p_{i_1} was delayed before time t_0. Namely, there is a triple $\langle p_{i_2}, p_{i_1}, t_1\rangle$ and no such triple for p_{i_1} in any time between t_1 and t_0. Continue the sequence in this way until reaching a packet p_{i_ℓ} was not delayed prior to time $t_{\ell-1}$.

It is convenient to define also $t_{-1} = T_{i_0}$ (the arrival time of p_{i_0}) and $t_\ell = 0$ (the start time of p_ℓ).

We get a sequence $\mathcal{DS}$ of triples

$$\mathcal{DS} \;=\; \langle p_{i_1}, p_{i_0}, t_0\rangle,\ \langle p_{i_2}, p_{i_1}, t_1\rangle,\ \ldots,\ \langle p_{i_\ell}, p_{i_{\ell-1}}, t_{\ell-1}\rangle,$$

where $T_{i_0} = t_{-1} > t_0 > t_1 > \ldots > t_{\ell-1} > t_\ell = \tau^A_{i_\ell} = 0$.

Lemma 4.5 $T_{i_0} \leq d_{i_0} + \ell$.

Proof: By definition of the relation C, we have the inequalities

(X_j) $\quad d_{i_{j+1}}(t_j) \leq d_{i_j}(t_j), \qquad$ for $j = 0, 1, \ldots, \ell - 1$.

For $j = 0, \ldots, \ell$ let θ_j denote the segment of the route ρ_{i_j} traversed by p_{i_j} between the times t_j and t_{j-1}, and let

$$\Delta_j = |\theta_j| = d_{i_j}(t_{j-1}) - d_{i_j}(t_j).$$

Substitute this definition in the inequalities (X_j) to get

(Y_j) $\quad d_{i_{j+1}}(t_{j+1}) + \Delta_{j+1} \leq d_{i_j}(t_j), \qquad$ for $j = 0, \ldots, \ell - 1$.

We also have

(Y_{-1}) $\quad d_{i_0}(t_0) + \Delta_0 = d_{i_0}(t_{-1}) = d_{i_0}$.

Summing the inequalities (Y_j) for $j = -1, 0, \ldots, \ell - 1$, we get

$$\Delta_0 + \Delta_1 + \ldots + \Delta_{\ell-1} + \Delta_\ell \leq d_{i_0} \qquad (2)$$

We also construct a chain of equalities for the times involved in these collisions. Since packet p_{i_j} (for $0 \leq j \leq \ell - 1$) was delayed at time t_j but never delayed since that time until time t_{j-1}, we have

(Z_j) $t_{j-1} = t_j + 1 + \Delta_j$, for $j = 0, \ldots, \ell - 1$.

We also have

(Z_ℓ) $t_{\ell-1} = d_{i_\ell}(t_{\ell-1}) = \Delta_\ell + t_\ell$.

Combining the equalities (Z_j) for $j = 0, \ldots, \ell$ we get

$$T_{i_0} = t_{-1} = \Delta_0 + \Delta_1 + \ldots + \Delta_{\ell-1} + \Delta_\ell + t_\ell + \ell. \tag{3}$$

Combining Eq. (2) and (3) with the assumption that $t_\ell = \tau^A_{i_\ell} = 0$, we get that

$$T_{i_0} \leq d_{i_0} + \ell,$$

and the lemma follows. ∎

In order to complete the proof of the theorem, it therefore remains to bound the length of the sequence $\mathcal{DS}$. This is done by proving the following claim.

Lemma 4.6 *The packets p_{i_j} appearing in the triples of the sequence $\mathcal{DS}$ are all distinct.*

Proof: The main idea of the proof is to show that if there is a packet that appears twice in $\mathcal{DS}$, then the path traverse by this packet is not the shortest. This fact is established by constructing an alternative path, using the route segments traversed by the packets in $\mathcal{DS}$, and showing that this alternative path is shorter.

The proof is by contradiction. Assume that some packet occurs twice in the sequence, for instance $p_{i_m} = p_{i_r}$ (or, $i_m = i_r$,) for $m > r$. By the structure of the sequence, every two consecutive packets are distinct, so necessarily $m \geq r + 2$. This means that the sequence contains a subcycle

$$\langle p_{i_{r+1}}, p_{i_r}, t_k \rangle,\ \langle p_{i_{r+2}}, p_{i_{r+1}}, t_{r+1} \rangle,\ \ldots,\ \langle p_{i_{m-1}}, p_{i_{m-2}}, t_{m-2} \rangle,\ \langle p_{i_m}, p_{i_{m-1}}, t_{m-1} \rangle = \langle p_{i_r}, p_{i_{m-1}}, t_{m-1} \rangle$$

where $t_r > t_{r+1} > \ldots > t_{m-1}$, and $m \geq r + 2$. (See Figure 1.)

We argue that among the inequalities (X_j), for $r \leq j \leq m - 1$, at least one of the inequalities is strict. Otherwise, all the collision resolutions in the cycle were made on the basis of indices, so $i_r = i_m < i_{m-1} < \ldots < i_{r+1} < i_r$; contradiction.

It follows that among the corresponding inequalities (Y_j), for $r \leq j \leq m - 2$, plus (X_{m-1}), at least one is strict.

Combine these inequalities in chain. Recalling that at least one of the inequalities is strict, we get

$$d_{i_m}(t_{m-1}) + \Delta_{m-1} + \ldots + \Delta_{r+1} < d_{i_r}(t_r). \tag{4}$$

Finally, let $\bar{\theta}$ denote the segment of the route ρ_{i_r} traversed by p_{i_r} between the times t_{m-1} (when it won) and t_r (when it lost), and let $\bar{\Delta} = |\bar{\theta}| = d_{i_r}(t_r) - d_{i_r}(t_{m-1})$. We get that

$$\Delta_{m-1} + \ldots + \Delta_{r+1} < d_{i_r}(t_r) - d_{i_r}(t_{m-1}) = \bar{\Delta},$$

or in other words, the segment $\bar{\theta}$ of ρ_{i_r} is not shortest; contradiction to the assumption that all the paths in $\mathcal{P}$ are shortest paths. ∎

Corollary 4.7 *The sequence $\mathcal{DS}$ is of length $\ell \leq k-1$.* ∎

Combining this corollary with Lemma 4.5 completes the proof of Part (1) of the theorem. Part (2) follows immediately. ∎

5 Greedy algorithms in the general case

The purpose of this section is to demonstrate the fact that, unlike the case of leveled routes, for general route classes not every greedy algorithm delivers the messages fast. Moreover, even the specific greedy algorithm used in Section 4 is not good enough in the general case. In fact, other "reasonable" greedy algorithms have bad performance too. Some of the complications of the following proof result from the fact that we strive to make it hold also for the case of "long" ($\Theta(|V|)$ routes and "many" ($\Theta(|V|)$ packets. We start with analyzing the following *fixed priority* algorithm.

Theorem 5.1 *There exist a graph G and a collection of paths $\mathcal{P}$ whose schedule time under the fixed priority algorithm is $T(\mathcal{P}) = \Omega(d\sqrt{k})$ even for $d, k = \Omega(|V|)$.*

Proof: Define the graph $G = (V, E)$ to be the union of the subgraphs $G^0, L^1, ..., L^x, C, S$, where x is a parameter to be determined later. See Figure 2. Intuitively, G^0 is the "main" route, while the purpose of the subgraphs L^i is to generate delays.

The nodes of the subgraph G^0 are

$$V(G^0) = \{1, \cdots, x\} \times \{1, \cdots, \frac{n}{x}\} \times \{\mathtt{in}, \mathtt{out}\},$$

and the edges are

$$\begin{aligned} E(G^0) &= \{(v_{<i,j,\mathtt{out}>}, v_{<i,j+1,\mathtt{in}>}) : 1 \le i \le \frac{n}{x}, 1 \le j \le x-1\} \cup \\ &\quad \{(v_{<i,j,\mathtt{in}>}, v_{<i,j,\mathtt{out}>}) : 1 \le i \le x, 1 \le j \le \frac{n}{x}\} \cup \\ &\quad \{(v_{<i,x,\mathtt{out}>}, v_{<i+1,1,\mathtt{in}>}) : 1 \le i \le \frac{n}{x}-1\}. \end{aligned}$$

The graph L is a straight line of y nodes and $y-1$ edges, i.e., $V(L) = \{l_1, \cdots, l_y\}$ and $E(L) = \{(l_i, l_{i+1}) : 1 \le i \le y-1\}$. We require that $y \ge x$, where the exact value of y will be determined later. For $1 \le i \le x$, the subgraph L^i is a copy of the graph L with superscript i.

The set C of edges connects nodes in G^0 to nodes in L^i. The first node in L^i, node l_1^i, is connected to a node $v_{<i,j,\mathtt{in}>}$, for $1 \le j \le \frac{n}{x}$. The last node in L^i, node l_y^i, is connected to the "next" node on G^0, that is, to node $v_{<i,j,\mathtt{out}>}$, for $1 \le j \le \frac{n}{x}$. Formally,

$$C = \{(v_{<i,j,\mathtt{in}>}, l_1^i) : 1 \le i \le x, 1 \le j \le \frac{n}{x}\} \cup \{(l_y^i, v_{<i,j,\mathtt{out}>}) : 1 \le i \le x, 1 \le j \le \frac{n}{x}\}$$

The component S includes the start node s, and one edge $(s, v_{<1,1,\mathtt{in}>})$.

When the algorithm starts, all $n = k$ packets are in node s. We denote the *identifier* of a packet by a pair of numbers $[i, j]$, where $1 \le i \le \frac{n}{y}$ and $1 \le j \le y$. We define a complete order between the packet identifiers by comparing them lexicographically. When comparing the identifiers of two packets, the packet with the smaller identifier has higher priority. The packets are grouped into *batches* according to their first identifier field, setting $B(i) = \{[i, j] \mid 1 \le j \le y\}$. The packets of each batch $B(i)$ must all traverse the same route ρ_i.

The routes of all the packets start with the edge $(s, v_{<1,1,\mathtt{in}>})$. The route ρ_1 (of batch $B(1)$) traverses L^1, returns to node $v_{<1,1,\mathtt{out}>}$, from there to $v_{<1,2,\mathtt{in}>}$ and traverses L^2, etc. Formally, define the edge sets

$$R_j = \bigcup_{1 \le i \le x} \left(\{(v_{<j,i,\mathtt{in}>}, l_1^i)\} \cup L^i \cup \{(l_y^i, v_{<j,i,\mathtt{out}>})\} \right)$$

and

$$M_j = \{(s, v_{<1,1,\mathtt{in}>})\} \cup \bigcup_{1 \le i \le x-1} \{(v_{<j,i,\mathtt{out}>}, v_{<j,i+1,\mathtt{in}>})\}.$$

Then ρ_j is composed of the edges of $M_1 \cup \cdots \cup M_{j-1} \cup R_j$.

Consider what happens until the packets of batch $B(1)$ arrive at their destination. Note that until time y, only packets of $B(1)$ move, since the others wait for them to be done with edge $(s, v_{<1,1,\mathtt{in}>})$.

At time $y+1$, the packet $[2,1]$ arrives at $v_{<1,1,\mathtt{in}>}$, and at time $y+2$ at node $v_{<1,1,\mathtt{out}>}$. However, at that time the first packet of $B(1)$, i.e., $[1,1]$, also arrives at node $v_{<1,1,\mathtt{out}>}$ (from node l^1_y). Thus, by time $2y+2$ (on which the first batch traverses the edge $(v_{<1,1,\mathtt{out}>}, v_{<1,2,\mathtt{in}>})$), all the packets of $B(2)$ are still delayed at node $v_{<1,1,\mathtt{out}>}$.

It takes packet $[2,1]$ $y+2$ time units to traverse the subpath from s to $v_{<1,1,\mathtt{out}>}$. The same argument as above can be repeated for every L^i. Therefore, when packet $[2,1]$ arrives at $v_{<2,1,\mathtt{in}>}$ the time is $x(y+2)$. At this time the packets of $B(1)$ arrive their destination. A similar argument as above shows that when the j^{th} batch has traversed $M_1, \cdots, M_{j-1}$, the time is $jx(y+2)$. Thus the total time is $\Omega(\frac{n}{y}xy)$. The theorem follows by choosing $x = y = \lfloor\sqrt{n}\rfloor$. (Note that $|V| = \Theta(n) = \Theta(k)$.)
∎

A similar, although somewhat more complicated proof can be constructed for other greedy algorithms, specifically the Min Went policy of Section 4, the analogous *Max Went* policy, or the *FIFO policy*, namely, the algorithm that resolves a collision between two packets in node v by sending the first to have arrived at node v (breaking ties by packet identifier numbers).

Theorem 5.2 *There exist a graph G and a collection of paths $\mathcal{P}$ whose schedule time under the the Min Went, Max Went and FIFO policies is $T(\mathcal{P}) = \Omega(d\sqrt{k})$ even for $d, k = \Omega(|V|)$.*

Proof: The proof is obtained by simple variations on the construction in the proof of Theorem 5.1.

For the Max went policy, observe that the schedule constructed in the proof of Theorem 5.1 obeys the Max Went policy.

For the FIFO policy we modify the graph, by changing the set C to

$$C = \{(v_{<i,j,\mathtt{in}>}, l^i_1) \,:\, 1 \le i \le x\} \cup \{(l^i_k, v_{<i,j,\mathtt{out}>}) \,:\, 1 \le i \le x, 1 \le k \le y\}.$$

The routes of the packets also change. The k^{th} packet in the j^{th} batch, i.e. packet $[k,j]$, moves from l^i_{y-k+1} to $v_{<i,j,\mathtt{out}>}$, for $1 \le i \le x$. Formally, let

$$R_{k,j} = \bigcup_{1 \le i \le x} \{(v_{<j,i,\mathtt{in}>}, l^i_1), (l^i_1, l^i_2), \ldots, (l^i_{y-k}, l^i_{y-k+1}), (l^i_{y-k+1}, v_{<j,i,\mathtt{out}>})\}$$

and now the route of packet $[k,j]$ is $M_1 \cup \ldots \cup M_{j-1} \cup R_{k,j}$.

Note that the whole first batch arrives at node $v_{1,1,\mathtt{out}}$ at the same time. Hence the FIFO policy too resolves the collisions (with the second batch) in favor of the first batch. Similarly, the resolution of the conflicts in this case is the same as before.

For the Min Went Policy the construction is similar but somewhat more complicated, due to the fact that the packets "switch priorities" during the execution; whenever a packet uses one of the bypasses L^i, its priority reduces. This requires us to organize the connections between the delay loops and the segments of G^0 in a more careful way, according to the priorities upon leaving the current segment. We omit the details of this construction. ∎

Acknowledgment

It is a pleasure to thank Amotz Bar-Noy for working with us in the early stages of this research.

References

[BGGJP85] A. E. Baratz, J.P. Gray, P.E. Green, J.M. Jaffe, and D.P. Pozenski, SNA Networks of Small Systems, *IEEE Trans. on Comm.* **sac-3**, May 1985, 416–426.

[BG87] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 1987.

[CGK88] Israel Cidon, Inder Gopal and Shay Kutten, New Models and Algorithms for Future Networks. *Proc. 7th Annual ACM Symp. on Principles of Distributed Computing*, Toronto, Canada, August 1988, 74–89.

[Ko78] Hisashi Kobayashi, *Modeling and Analysis*, Addison-Wesley, 1978.

[LMR88] T. Leighton, B. Maggs, and S. Rao, Universal Packet Routing Algorithms, *Proc. 29th IEEE Symp. on Foundations of Computer Science*, White Plains, NY, October 1988, 256–269.

[Mar82] James Martin, *SNA: IBM's Networking Solution*, Prantice Hall, Englewood Cliffs, NJ, 1982

[MRR80] J. McQuillan, I. Richer and E.C. Rosen, The New Routing Algorithm for the ARPANET, *IEEE Trans. on Commun.* **com-28**, May 1980, 711–719.

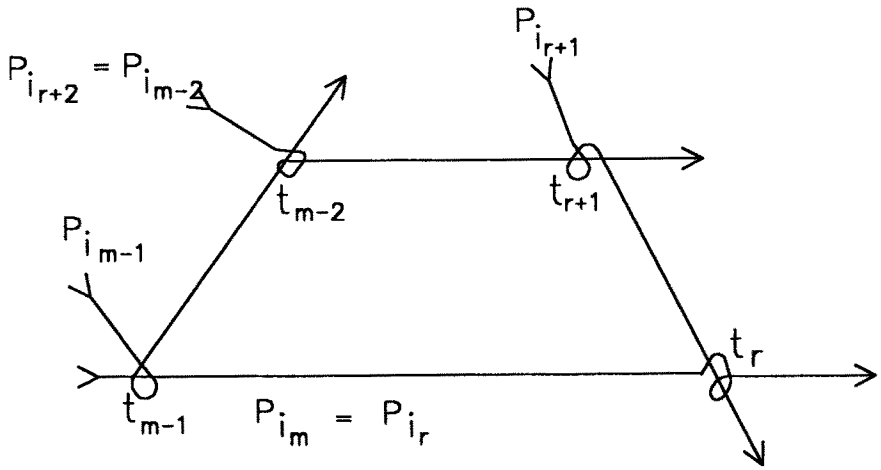

Figure 1 - Time Ordered Cycle

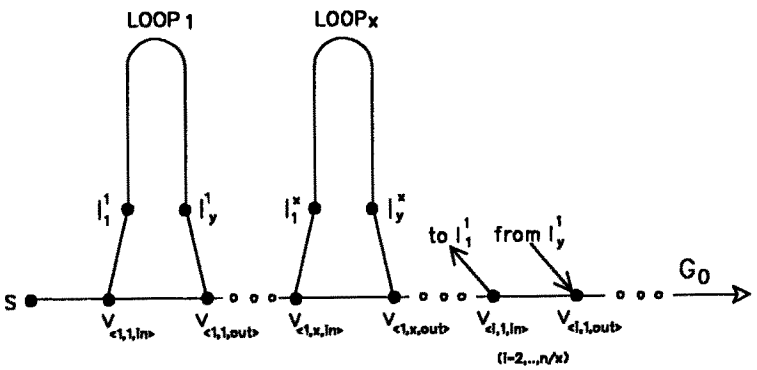

Figure 2 - Counter Example for Arbitrary Routes