JOURNAL OF ALGORITHMS 14, 449–465 (1993)

Greedy Packet Scheduling on Shortest Paths*

YISHAY MANSOUR[†]

IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598

AND

BOAZ PATT-SHAMIR[‡]

Laboratory for Computer Science, MIT, Cambridge, Massachusetts 02139

Received July 25, 1991; revised April 1992

We investigate the simple class of greedy scheduling algorithms, that is, algorithms that always forward a packet if they can. Assuming that only one packet can be delivered over a link in a single step and that the routes traversed by a set of packets are distance optimal ("shortest paths"), we prove that the time required to complete transmission of a packet in the set is bounded by its route length plus the number of other packets in the set. This bound holds for *any* greedy algorithm, even in the case of different starting times and different route lengths. The bound also generalizes, in the natural way, to the case in which w packets may cross a link simultaneously. Furthermore, the result holds in the asynchronous model, using the same proof technique. The generality of our result is demonstrated by a few applications. We present a simple protocol, for which we derive a general bound on the throughput with any greedy scheduling. Another protocol for the dynamic case is presented, whose packet delivery time is bounded by the length of the route of the packet plus the number of packets in the network in the time it is sent.
© 1993 Academic Press, Inc.

1. INTRODUCTION

The performance of a routing protocol is evaluated by various measures, such as global network throughput, maximum packet delivery time, storage

*A preliminary version of this paper appears in the "Proceedings of 10th Annual ACM Symposium on Principles of Distributed Computing."

[†]Part of the research was done while the author was at Aiken Computation Laboratory, Harvard University, and supported in part by ONR under Contract N00014-85-K-0445.

[‡]Supported in part by DARPA under Contract N00014-87-K-0825.

requirement etc. Conceptually, we may break the task of routing a set of packets across a communication network into two subproblems, which we refer to as *path selection* and *queuing policy*. The path selection determines the routes that packets traverse, while the queuing policy determines which of the packets currently in the queue at a node to forward over the link.

Specifically, the path selection problem is defined as follows. Given the network topology and the locations of the sources and the destinations of a set of packets, select the transmission paths. The selection may be made at the destination, as in circuit switching networks [4], or while the packet progresses in the network, as is common in packet switching networks [5].

The need for a queuing policy emerges from the following physical restriction of the network. The *width* of the links is bounded, i.e., only w packets can progress over a link at a time, where w is some positive integer. However, it is conceivable that more than w packets wish to traverse the same link simultaneously. The task of a queuing policy is to schedule the pending packets over the link. In practice, queuing policies exhibit assorted combinations of FIFO, fixed priorities, some sort of congestion control, etc. (see [1, 7]). In most cases the queuing policy is *greedy*; that is, a packet is delayed only if w other packets currently progress over the link.

The isolation of path selection and queuing policy allows us to study the effects and relations between them. In this paper we bound the delivery time for the case in which the paths are shortest and the schedule is greedy. We stress that our results do not depend on the separation between the two subtasks.

Most of the related previous work concentrated on routing in a synchronous model and assumed that $w = 1$. The abstraction of routing protocols, as presented here, was first studied by Leighton, Maggs, and Rao [3]. In this abstraction, there is a fixed set of k packets, and a route is assigned to each packet. The packets traverse the assigned routes. We study the time required for delivering a packet, where the time for a packet to traverse any link is at most one time unit.

Leighton *et al.* [3] proved that there exists a schedule that delivers all packets in $O(d + c)$ time, where d is the maximal path length and c is an upper bound on the number of routes that share a single edge. Unfortunately, the proof is non-constructive, and no polynomial time algorithm to compute this optimal schedule is known. They also provide two randomized distributed protocols for the problem. The first applies to arbitrary sets of paths and requires $O(c + d\log|V|)$ time. The second protocol applies to the case of *l-leveled* paths (where the paths can be partitioned to l levels) and completes the packet delivery in $O(c + l + \log|V|)$ time. The latter protocol requires all packets to start at the same time.

Cidon *et al.* [2] conjectured that in the synchronous model, if the path selection algorithm produces shortest paths, then any greedy queuing policy delivers all k packets within $d + k - 1$ time units, assuming $w = 1$. Previously, only special cases of this general conjecture were proved. In their classical work on randomized routing on a hypercube, Valiant and Brebner [8] show, as one of the steps in the proof, that any greedy scheduling of k packets on a line of d processors requires no more than $d + k - 1$ time units. Cidon *et al.* proved their conjecture in the special case of leveled paths and showed that this bound holds for arbitrary shortest paths when coupled with a specific greedy queuing policy, denoted by Min-Went (the queuing policy at the nodes is to first forward packets that traversed the least distance so far).

Rivera-Vega *et al.* [6] considered the related problem of "file redistribution scheduling." They investigated various computational aspects of the problem. In one of their results they prove that scheduling k packets over shortest paths of length d, using fixed-priority policy, takes at most $d + k - 1$ time units. The result holds in the general case in which start times may be different, and the time bound for a packet does not depend on the length of other paths.

In [2] it is also shown that if an arbitrary set of routes is allowed, then many natural routing strategies (e.g., priority, FIFO, Min-Went) may require $\Omega(n^{1.5})$ time, when d and k are $\Theta(n)$. This result suggests yet another motivation for considering shortest paths.

In this work we settle the conjecture posed in [2] in the affirmative. Specifically, we consider the general scenario in which k packets traverse shortest paths and the queuing policy is *any* greedy queuing policy. We show that any such schedule is guaranteed to deliver packet p_i in no more than $d_i + \lfloor (k - 1)/w \rfloor$ time units, where d_i is the number of links p_i traverses, and w is the number of packets that can traverse a link simultaneously. Our proof technique extends to the asynchronous case, yielding the same bound.

Our result is obtained for the most general case, where the routes may have different lengths and the packets are allowed to start traversing the network at arbitrary time. This generality enables us to derive a few interesting applications in the *dynamic* model, in which the total number of packets is not bounded and packets are continuously generated and delivered. Specifically, we consider the *throughput* of the network. Let n be the number of nodes in the network, and assume that each node has an arbitrarily large number of packets to send. We present a simple congestion control protocol that guarantees, for any greedy queuing policy, even in the asynchronous model, that every $O(n)$ time units $\Omega(nw)$ packets are delivered. The congestion control protocol resides in the source nodes, and its task is to decide when the next packet will be sent. We also show that if one selects a greedy queuing policy in which a packet is never delayed by packets generated after it was generated, then each packet p_i arrives at its destination within $d_i + n$ time units.

For simplicity of presentation, we first prove our results in the synchronous setting, and later discuss the asynchronous case. In Section 2 we define formally the problem in the synchronous model. In Section 3 we state and prove our main result in the synchronous model. Applications of the main result in a dynamic network are presented in Section 4. In Section 5 we describe the problem in the asynchronous model and prove our main result in this model. In Section 6 we make a few concluding remarks and point to some open problems.

2. The Synchronous Model

In this section we define the routing problem and its relevant parameters. We model the communication network as a directed graph $G = (V, E)$, where an edge (u, v) represents a unidirectional link from processor u to v. There is a collection of k packets p_i and k associated simple paths, referred to as *routes*, $R_i = (v_0^i, \ldots, v_{d_i}^i)$, $1 \le i \le k$. Packet p_i is transmitted along route R_i whose length is d_i. We deal with shortest path routes, i.e., d_i is equal to the distance from the origin to the destination of R_i for all $1 \le i \le k$.

In the synchronous communication model we assume the existence of a global clock, characterized by the property that a packet sent at time t is received by time $t + 1$. Packet p_i starts traversing its route at time t_s^i and is delivered at time t_e^i. The *duration* of packet p_i is the time interval $[t_s^i, t_e^i]$ in which p_i is in transit. A *schedule* is a mapping S of a packet and a time step to a node in the graph. Intuitively, the schedule describes the location of any packet at all time steps. At a given time step, a packet may either progress along its route or remain in the queue. Therefore, the schedule must satisfy the following condition. For all $1 \le i \le k$ and time steps $t_s^i \le t < t_e^i$, either packet p_i *does not progress at time* t, i.e., $S(p_i, t + 1) = S(p_i, t)$, or p_i *progresses at time* t; i.e., $(S(p_i, t), S(p_i, t + 1))$ is an edge in R_i. The nodes $S(p_i, t_s^i) = v_0^i$ and $S(p_i, t_e^i) = v_{d_i}^i$ are the *source* and the *destination* of p_i, respectively.

Packets p_i and p_j are said to *meet* at time t if $S(p_i, t) = S(p_j, t)$. In this paper we consider only schedules that satisfy the *link width condition*. This condition is that at most w packets may progress over the same link at a single time step. We call the parameter w the *width* of the links.

Packet p_i is said to *delay* packet p_j at time t_0 if p_j does not progress at time t_0 and the next link in the route of p_j is currently traversed by p_i.

Formally, at time t_0 packet p_i meets p_j and progresses, and $(S(p_i, t_0), S(p_i, t_0 + 1))$ is an edge in R_j, the route of p_j.

A *greedy schedule* is a schedule such that if there are q packets waiting to be forwarded on some link, then $\min(q, w)$ of these packets are forwarded. Formally, if p_j does not progress at time t_0, then there exist w packets that delay p_j at time t_0.

In Section 5 we generalize the above definitions to the asynchronous case.

3. Greedy Schedules for Shortest Paths

In this section we consider greedy scheduling in the synchronous model for *shortest paths*, i.e., where the length of each route is equal to the distance from its origin to destination. Our result is summarized in the following theorem.

THEOREM 3.1. *Let $p_1, \dots, p_k$ be a set of packets whose routes are $R_1, \dots, R_k$, respectively. Suppose each R_i is a shortest path of length d_i, $1 \leq i \leq k$. Then in any greedy schedule each packet p_i arrives at its destination within $d_i + \lfloor (k-1)/w \rfloor$ time units, where w is the width of the links.*

In the rest of the section we fix a greedy schedule S. Since we concentrate on a packet p_i, we consider only the time interval $[t_s^i, t_e^i]$ in which p_i traverses R_i (the interval is finite since trivially $t_e^i \leq t_s^i + k \cdot \max_j\{d_j\}$).

The main idea of the proof is based on the concept of *time path*. To illuminate this concept we suggest the following analogy. Picture a train conductor, who boards a certain train early in the morning, on which he gets his lunch at noon. (The trains are analogous to packets, and the conductor's tour is analogous to the time path.) Each train travels a known route, and the conductor has the train schedule by which he can infer which train delays the other at intersections. (Analogously, our analysis is on a given fixed schedule, from which we can infer which packet delays which.) The conductor wishes to inspect as many trains as possible. Naturally, he can change trains only whenever his train meets another train. However, he is not willing to give up his lunch, which is served on the first train he has boarded. For this reason, the conductor carefully plans his tour in advance. What we would like to show is that the conductor can find a tour in which he is delayed by each train at most once and returns to his original train on time. Clearly, the time that the conductor travels is the same as the time of the lunch train, and therefore we can limit our interest to bounding the travel time of the conductor.

The concept of time path is formalized as follows.

DEFINITION 3.2. Let $t_s \leq t_e$ be time steps. A function τ that maps a time steps interval $[t_s, t_e]$ to packets is a *time path* if $S(\tau(t), t) = S(\tau(t-1), t)$ for all time steps $t_s < t \leq t_e$. The *location* of a time path τ at time t, denoted $L(\tau, t)$, is defined as $S(\tau(t), t)$. The *trace* of the τ is the sequence $L(\tau, t_s), \dots, L(\tau, t_e)$, after deleting repetitions. The time interval $[t_s, t_e]$ is the *duration* of τ.

From the definition of a time path it follows that the trace of a time path is a path in the graph. Another simple observation is that mapping all the time steps to the same packet is a time path.

Informally, the outline of the proof of Theorem 3.1 is as follows. Given a specific packet p_i, we define an initial time path (whose duration is $[t_s^i, t_e^i]$, the time in which p_i is in transit) that maps all time steps to p_i. Then we manipulate this time path in a way such that the length of its trace remains invariant. We further show that a time path can be modified until the resulting time path is delayed at most once by each packet and is not delayed by its last packet. Thus, each time in which the final time path is delayed, it is delayed by w different packets. This, in conjunction with the fact that the trace of the final time path has the same length the original route has, proves that the time for this packet is bounded by $d_i + \lfloor (k-1)/w \rfloor$.

We start with some additional definitions concerning time paths. For a time path τ and a time step t, we say that τ is *delayed* at time t if $L(\tau, t) = L(\tau, t+1)$; τ is delayed by a packet p, where p is one of the w packets that delayed packet $\tau(t)$ at time t. The existence of w such packets is guaranteed by the greedy nature of the schedule.

The following abstract operation is our tool for manipulating time paths.

DEFINITION 3.3. *Switch* is a mapping from time paths to time paths. *Switch* is *applicable* to a time path τ if there exists a packet that delays τ at some time t and meets τ at some time $t' > t$. Suppose *Switch* is applicable to τ. Let p be the first packet that delays τ and later meets τ. Let t_0 be the first time p delays τ, and t_1 the first time after t_0 that p meets τ again. For all $t_s \leq t \leq t_e$, Switch$(\tau)(t)$ is defined as follows.

$$\text{Switch}(\tau)(t) = \begin{cases} p, & \text{for } t_0 \leq t < t_1 \\ \tau(t), & \text{otherwise.} \end{cases}$$

The packet p is called the *detour packet*, the time interval $[t_0, t_1)$ is called the *detour time*, and t_0 is called the *rank* of τ.

Carrying on with the analogy to our train conductor, we say that the *Switch* operation is a tool by which he can safely switch trains and be on time for lunch, since whenever he switches trains he is guaranteed to return to his previous tour. The conductor's strategy in planning his tour is as follows. His initial tour consists merely of the lunch train; he then repeatedly applies *Switch* to his current tour, until *Switch* is not applicable anymore. We will prove that this strategy always terminates.

The rest of this section is organized as follows. First, we prove that the *Switch* operation preserves the source, destination, and duration of a time path. Next, we show that *Switch* also preserves the shortest path property. Finally, we show that *Switch* cannot be applied iteratively infinitely many times on *any* time path.

Note that once we reach a time path to which *Switch* is not applicable, we are guaranteed that this time path is delayed at most once by each packet. The result follows, since the duration of a time path is bounded by the length of its trace plus the number of times it is delayed.

In the following lemma we establish some immediate properties of time paths and the *Switch* operation.

LEMMA 3.4. *Let τ be a time path with duration $[t_s, t_e]$. Assume that* Switch *is applicable to τ, and let t_0 be the rank of τ. Denote τ' =* Switch(τ). *Then the following properties hold.*

1. *τ' is a time path with duration $[t_s, t_e]$.*

2. *τ' is not delayed at time t_0.*

3. *The source and the destination of the traces of τ and τ' are identical; i.e., $L(\tau', t_s) = L(\tau, t_s)$, and $L(\tau', t_e) = L(\tau, t_e)$.*

4. *Suppose that p and $\tau(t_e)$ meet at time t_e. If p delays τ at time $t_s \leq t < t_e$ then* Switch *is applicable to τ.*

Proof. 1. Obviously, τ' is defined in the time interval $[t_s, t_e]$. Let $[t_0, t_1)$ be the the detour time, and let p be the detour packet. By the definition of *Switch*, τ and τ' are identical in the two time intervals $[t_s, t_0)$ and $[t_1, t_e]$, and $\tau'(t) = p$ for all $t_0 \leq t < t_1$. Hence $S(\tau'(t), t-1) = L(\tau', t-1)$ for all time steps $t_s < t \leq t_e$ except for $t = t_0$ or $t = t_1$. Suppose now that $\tau(t_0 - 1) = p_0$. Since p delays p_0 at time t_0, we have that $S(p_0, t_0) = S(p, t_0)$, which implies $S(\tau'(t_0 - 1), t_0) = L(\tau', t_0)$. Similarly, the condition holds for $t = t_1$.

2. Let p be the detour packet. By definition of *Switch*, $\tau'(t_0) = p$. Since *Switch* is applicable to τ, packet p delays τ at time t_0. That is, $S(p, t_0) = L(\tau, t_0) = L(\tau, t_0 + 1)$, and $S(p, t_0) \neq S(p, t_0 + 1)$. By part 1 above, τ' is a time path, and hence $L(\tau', t_0 + 1) = S(\tau'(t_0), t_0 + 1)$. Since $\tau'(t_0) = p$, we have that $L(\tau', t_0) \neq L(\tau', t_0 + 1)$, which implies that τ' is not delayed at time t_0.

3. Consider the destination first. By the definition of *Switch*, $\tau'(t_e) = \tau(t_e)$, and hence $L(\tau', t_e) = L(\tau, t_e)$. As for the source, let t_0 denote the

rank of τ, i.e., the first time in which τ' differs from τ. If $t_0 > t_s$ then clearly $L(\tau', t_s) = L(\tau, t_s)$. Otherwise, the detour packet p delayed τ at time t_s, and hence $L(\tau', t_s) = S(p, t_s) = L(\tau, t_s)$.

4. Immediate from the definition of *Switch* applicability. □

Next, we show that *Switch* preserves the shortest path property.

LEMMA 3.5. *Suppose* Switch *is applicable to time path τ, and let p be the detour packet. If the trace of τ is a shortest path and the route of p is a shortest path, then the trace of* Switch(τ) *is a shortest path.*

Proof. Let $[t_0, t_1)$ be the detour time. Denote $v = L(\tau, t_0)$ and $u = L(\tau, t_1)$. Consider the following two path segments. The first is the segment of the trace of τ induced between t_0 and t_1, and the second is the segment of the route of p connecting v and u (the fact that τ' is a time path implies that $S(p, t_0) = v$ and $S(p, t_1) = u$). Since both segments are subpaths of shortest paths, they are the shortest paths between their endpoints, and since they have the same endpoints, their lengths are equal. Therefore, the length of the trace of *Switch*(τ) is equal to the length of the trace of τ. Having this fact, together with the fact that the traces of τ and *Switch*(τ) share the same endpoints (Lemma 3.4, part 3), and with the assumption that the trace of τ is a shortest path, we conclude that the trace induced by *Switch*(τ) is a shortest path, too. □

Having proved the above lemmas for a single application of *Switch* to a time path, we turn to deal with iterative applications of *Switch*.

DEFINITION 3.6. A sequence $\tau_0, \tau_1, \ldots$ is a *time path sequence* if τ_0 is a time path, and $\tau_{j+1} = \text{Switch}(\tau_j)$ for all $j \geq 0$.

The following lemma establishes the fact that between two applications of *Switch* with rank r there must be a *Switch* operation with rank *strictly* smaller than r.

LEMMA 3.7. *Let $\tau_0, \tau_1, \ldots$ be a time path sequence. Suppose that the rank of τ_i is r, and assume that the rank of τ_j is also r, where $i < j$. Then there exists $i < l < j$ such that the rank of τ_l is strictly smaller than r.*

Proof. By the definition of *Switch*, the rank of a time path τ must be equal to some time step in which τ is delayed. Assume, for contradiction, that the rank of τ_l, for all $i < l < j$, are at least r. Note that if the rank of τ_{l+1} is at least r, and τ_l is not delayed at time r, then rank of τ_{l+1} is at least $r + 1$.

By part 2 of Lemma 3.4, τ_{i+1} is not delayed at time r. Hence the rank of τ_l must be strictly greater than r for all $i < l < j$. This implies that for all $i < l < j$, $\tau_l(t) = \tau_{l+1}(t)$ for all $t \leq r$; i.e., all these time paths have the same initial segment. This, in particular, implies that τ_j is not delayed at time r, a contradiction to the assumption that the rank of τ_j is r. □

We now prove the crucial property we need: *Switch* can be applied only a finite number of times.

LEMMA 3.8. *Given a schedule of a finite set of finite paths, all time path sequences are finite.*

Proof. First, note that the total number of time paths for the given set of paths is bounded (a trivial bound is k^{kd}, where k is a bound on the number of packets and d is a bound on the length of the routes). Suppose now, for contradiction, that there exists an infinite time path sequence. Then there must exist a cyclic time path sequence, i.e., time paths $\tau_j, \tau_{j+1}, \dots, \tau_{j+c}$ such that $\tau_j = \tau_{j+c}$, and $\tau_{i+1} = Switch(\tau_i)$ for all $j \leq i < j + c$. Let r be the minimal rank of the time paths in the cycle and suppose that the rank of τ_m is r. By the cyclicity, the rank of τ_{m+c} is also r. Hence, by Lemma 3.7, there must be $0 < l < c$, such that the rank of τ_{m+l} is strictly less than r, a contradiction to the minimality of r. □

We can finally prove our main result. Let us first restate it.

THEOREM 3.1. *Let $p_1, \dots, p_k$ be a set of packets whose routes are $R_1, \dots, R_k$, respectively. Suppose each R_i is a shortest path of length d_i, $1 \leq i \leq k$. Then in any greedy schedule each packet p_i arrives at its destination within $d_i + \lfloor (k-1)/w \rfloor$ time units, where w is the width of the links.*

Proof. Let p_i be any packet. Suppose that p_i is in transit on its route in the time interval $[t_s, t_e]$. Consider the time path sequence $\tau_0, \tau_1, \dots$ obtained by letting $\tau_0(t) = p_i$ for all time steps $t_s \leq t \leq t_e$, and letting $\tau_{i+1} = Switch(\tau_i)$. By Lemma 3.8, this sequence is finite. Let τ^* be the last time path in the sequence. The fact that *Switch* is not applicable to τ^* implies that no packet delays τ^* twice. Since each delay of τ^* is incurred by w different packets, and adding the fact that p_i does not delay τ^* (by part 4 of Lemma 3.4), we have that the total number of delays in τ^* is at most $\lfloor (k-1)/w \rfloor$, which implies that the duration of τ^* is at most its trace length plus $\lfloor (k-1)/w \rfloor$. From Lemma 3.5 it follows that the lengths of the traces of all the time paths in the sequence are equal, and since their duration is (by definition) unchanged, we conclude that p_i arrives at its destination in at most $d_i + \lfloor (k-1)/w \rfloor$ time steps. □

As a final remark we point out that in the proof we do not make use of the assumption that the routes are pre-determined. In fact, we only need to assume that the paths that were actually traversed are shortest.

4. Applications for Dynamic Networks

Theorem 3.1 provides an interesting insight into the relationship between queuing policies and the path selection schemes. In this section we give a few simple applications of the result in the *dynamic model*, in which packets are continuously generated by the nodes.

4.1. *Throughput of Networks*

In order to measure throughput of a network we assume that the nodes have an unbounded number of packets to send. The throughput of a network, intuitively, is the rate at which packets are delivered. We apply the result for the worst case delivery time of a single packet, obtaining bounds on throughput.

We define a *congestion control* protocol, i.e., a protocol that resides in the source nodes, whose task is to decide when to send the next packet. Consider the following generic protocol. When a packet is received by its destination node, an acknowledgment is sent back to the source. A new packet is sent only after an acknowledgment for a previous packet is received, in a way such that exactly w packets and acknowledgments are in transit to and from each node at any point in time. We assume that the packets and the acknowledgments traverse shortest paths.

Let n be the number of nodes in the network and assume that each node has an arbitrarily large number of packets to send. We show that every $O(n)$ time units nw packets are delivered, so long as the queuing policy is greedy.

The following lemma bounds the progress that is made in the network using the above congestion control protocol.

Lemma 4.1. *In any $3n$ consecutive time units, at least nw messages (either regular packets or acknowledgments) are delivered.*

Proof. As mentioned above, there are always nw messages in transit in the network. Let t be any point in time, and let $m_1, \dots, m_{nw}$ be the messages in transit at time t. If all these messages are delivered by time $t + 3n$, we are done. Otherwise, let m' be a message that is not delivered by time $t + 3n$, i.e., m' is delayed at least $2n$ times in the time interval $[t, t + 3n]$. Hence there are at least $2nw$ messages in transit in this time interval, and therefore, at least nw messages are delivered (again, since in every point in time there are only nw messages in transit). □

Using the bound on the number of messages delivered, we derive a bound on the number of "regular" packets delivered by the protocol, which is the throughput of the network.

THEOREM 4.2. *Consider the generic protocol, coupled with any greedy queuing policy. Then in $\Theta(n)$ consecutive time units, $\Omega(nw)$ packets are delivered.*

Proof. By the nature of the protocol, the type of the message associated with a node alternates between "regular packet" and "acknowledgment"—when a packet is delivered an acknowledgment is sent, and when an acknowledgment is received, a packet is sent. Hence, for every node, the difference between the number of packets delivered and the number of acknowledgments received, in any time interval, is at most w.

Consider a time interval of length $9n$. We show that in any such interval, at least nw packets are delivered. By Lemma 4.1, at least $3nw$ messages are delivered in any such time interval. If less than nw packets are delivered, then, by the pigeonhole principle, there must exist a node for which the number of received acknowledgments is greater than the number of delivered packets by at least $w + 1$, a contradiction. □

We remark that when the length of the time interval tends to infinity, a simple extension of the argument above shows that the average number of packets delivered per time unit is at least $w/2$.

Theorem 4.2 applies to any greedy queuing policy when coupled with the above congestion control protocol. However, Theorem 4.2 does not bound the delivery time of any single packet. Indeed, it might be the case that some of the packets will never reach their destinations. Several queuing policies overcome this undesirable behavior. For example, in the FIFO queuing policy, a packet is delayed $O(n^2/w)$ time units in the worst case. In Section 4.2 we show how, with the aid of a global clock, an extremely simple protocol gives a $O(n/w)$ bound on the number of delays per packet.

4.2. *Delivery Time*

We define the *delivery time* of a packet to be the time since the packet was generated until it is delivered. We also assume that there is a global clock accessible by all nodes. In the following theorem, we present a specific greedy queuing policy using a clock and such that the delivery time of a packet does not depend on packets generated after it.

THEOREM 4.3. *In a dynamic network with links of width w, if the packets traverse shortest paths, then there is a queuing policy that delivers any packet p within $d_p + \lfloor l_p/w \rfloor$ time units, where d_p is the length of the route of p, and l_p is the number of packets in the network when the p was generated.*

Proof. Our strategy is to define a queuing policy in which a packet p can be delayed only by packets in transit when p is generated, and to show that the theorem holds for the resulting schedule. Specifically, we employ a *priority* queuing policy, in which scheduling conflicts are resolved by a global fixed priority ordering. Each packet is assigned a priority, and if a scheduling conflict arises, then the packet with the higher priority is forwarded (ties may be broken arbitrarily).

Formally, the priority is a mapping $\mathscr{H}$ of packets to integers. The priority queuing policy is a greedy schedule that observes the rule that packet p delays p' only if they meet and $\mathscr{H}(p) \leq \mathscr{H}(p')$. We call the set $E_p = \{p': \mathscr{H}(p') \leq \mathscr{H}(p)\}$ the set of *effective packets* with respect to p, and denote $k_p = |E_p|$.

Given a priority queuing policy, observe that from the point of view of a packet p it seems as if only packets in E_p exist, since other packets do not influence its schedule. In particular, the schedule of the packets in E_p would not change even if all packets not in E_p are removed from the initial set of packets. Therefore, when analyzing the delivery time of p, we may assign $k = k_p$.

Consider the priority queuing policy where the time in which a packet is generated is assigned to be its priority. Since only packets that were in the network when this packet was generated may delay it, the discussion above implies the result. □

Combining the protocols described in Theorems 4.2 and 4.3, we obtain the following immediate corollary.

COROLLARY 4.4. *In a dynamic network with w-width links, if the packets traverse shortest paths, then there is a queuing policy such that*

1. *Each packet p is delivered within $d_p + n$ time units, where d_p is the length of the route of p.*
2. *In $\Theta(n)$ consecutive time units, $\Omega(nw)$ packets are delivered.*

5. SCHEDULES IN THE ASYNCHRONOUS SETTING

In this section we extend the proof of Theorem 3.1 to the asynchronous model. In Section 5.1 we define formally the problem in the asynchronous model. In Section 5.2 we state our result and prove it in this model. We remark that the proof is essentially the same as in Section 3.

5.1. *The Asynchronous Model*

As in the synchronous model, we model the communication network as a directed graph, where an edge (u, v) represents a unidirectional link from processor u to v. There is a collection of k packets p_i and k associated simple paths, referred to as *routes*, $R_i = (v_0^i, \dots, v_{d_i}^i)$, $1 \leq i \leq k$. Packet p_i is transmitted along route R_i whose length is d_i. We deal with shortest path routes, i.e., d_i is equal to the distance from the origin to the destination of R_i for all $1 \leq i \leq k$.

We denote the route of packet p_i by $R_i = v_0^i, \dots, v_{d_i}^i$. We associate with each packet p_i the time in which the *send events* $s_0^i, \dots, s_{d_i-1}^i$ and the *receive events* $r_1^i, \dots, r_{d_i}^i$ occur. Time s_j^i is the time in which packet p_i is sent from node v_j^i, and r_{j+1}^i is the time in which p_i is received in the other end of the link, i.e., at node v_{j+1}^i. For any schedule, the time of the events must satisfy $r_j^i \leq s_j^i < r_{j+1}^i$ for all $1 \leq i \leq k$ and $0 \leq j < d_i$. For the purpose of time analysis, we assume that a packet is in transit over a link at most one time unit, i.e., $0 < r_{j+1}^i - s_j^i \leq 1$ for $1 \leq i \leq k$ and $0 \leq j < d_i$.

A key difficulty in the asynchronous model is formalizing the intuitive notion of "previous time step" and "next time step," which in the synchronous model are simply $t - 1$ and $t + 1$, respectively. The limit operator allows us to present an elegant formulation of these notions in a continuous time system. The limit operator is used below in the definitions of schedule and time paths.

The schedule, as in the synchronous case, is a mapping from time to nodes. While a packet progresses over a link, the schedule maps it to the sending node. Formally, the schedule mapping is defined in the *continuous* interval $[s_0, r_{d_i}]$ by $S(p_i, t) = v_j^i$, where j is such that $r_j^i \leq t < r_{j+1}^i$ is satisfied. We may think of a schedule as right-continuous step function:

$$\lim_{\substack{t \to t_0 \\ t \geq t_0}} S(p, t) = S(p, t_0).$$

We fix, as in the synchronous case, the schedule and its related sequences of events. Packet p_i is said to *progress* at the time intervals $[s_j^i, r_{j+1}^i)$ for $0 \leq j \leq d_i$. The definition of *meet* remains unchanged: packet p_i meets p_j at time t if $S(p_i, t) = S(p_j, t)$.

Recall that the link width condition is that for every link, at any point in time, there are at most w packets in transit over the link. We say that packet p_i *delays* packet $p_{i'}$ in the time interval $[t_0, t_1)$, if during this time interval packet p_i progresses over some link (v, u), and $p_{i'}$ is in node v, waiting to be forwarded over (v, u). Formally, p_i delays $p_{i'}$ in the time interval $[t_0, t_1)$ if there exist j and j' such that $(v_j^i, v_{j+1}^i) = (v_{j'}^{i'}, v_{j'+1}^{i'})$, and $[s_j^i, r_{j+1}^i) \cap [r_{j'}^{i'}, s_{j'}^{i'}) \supseteq [t_0, t_1)$.

A queue policy is *greedy* if in its associated schedules a packet does not progress only if there exist w packets that delay it.

5.2. *Asynchronous Time Paths*

We are now ready to state and prove our main result in the asynchronous case. First, we formalize the definition of time path in the continuous fashion.

DEFINITION 5.1. Let $t_s \le t_e$ be time points. A function τ that maps the time interval $[t_s, t_e]$ to packets is a *time path* if for all times $t_s < t' \le t_e$,

$$\lim_{\substack{t \to t' \\ t < t'}} S(\tau(t), t') = S(\tau(t'), t').$$

The definitions of *location*, *trace*, and *duration* of a time path remain the same. We say that a packet p *delays* (resp., *meets*) a time path τ at time t if p delays (resp., meets) the packet $\tau(t)$ at time t. The definition of *Switch* requires no change under the revised definitions of a time path and the notion of a packet delaying another. We repeat it for the sake of completeness.

DEFINITION 5.2. *Switch* is a mapping from time paths to time paths. *Switch* is *applicable* to a time path τ if there exists a packet that delays τ at some time t, and it meets τ at some time $t' > t$. If *Switch* is applicable to τ, let p be the first such packet, let t_0 be the first time p delays τ, and let t_1 be the first time after t_0 that p meets τ again. For $t \in [t_s, t_e]$, Switch$(\tau)(t)$ is defined as follows:

$$\text{Switch}(\tau)(t) = \begin{cases} p, & \text{for } t \in [t_0, t_1) \\ \tau(t), & \text{otherwise.} \end{cases}$$

The packet p is called the *detour packet*, the time interval $[t_0, t_1)$ is called the *detour time*, and t_0 is called the *rank* of τ.

Note that if the detour interval is $[t_0, t_1)$, then times t_0 and t_1 are some *receive* or *send* event, and hence, given a schedule, there are at most $2\Sigma_i d_i$ possible ranks.

The proof of the asynchronous case proceeds along the same lines of the proof for the synchronous case. In what follows, for each claim we indicate the corresponding one in Section 3. The differences are minor.

The lemma below is analogous to Lemma 3.4. The proof uses the same arguments as in the proof of Lemma 3.4 and is therefore omitted.

LEMMA 5.3. *Let τ be a time path with duration $[t_s, t_e]$. Assume that* Switch *is applicable to τ, and let t_0 be the rank of τ. Denote $\tau' =$* Switch(τ). *Then the following properties hold.*

1. *τ' is a time path with duration $[t_s, t_e]$.*
2. *τ' is not delayed at time t_0.*
3. *$L(\tau', t_s) = L(\tau, t_s)$, and $L(\tau', t_e) = L(\tau, t_e)$.*
4. *Suppose that p and $\tau(t_e)$ meet at time t_e. If p delays τ at time $t_s < t < t_e$ then* Switch *is applicable to τ.*

The lemma below is analogous to Lemma 3.5. Its proof is identical to the proof of Lemma 3.5.

LEMMA 5.4. *Assume that* Switch *is applicable to time path τ, and let p be the detour packet. If the trace of τ is a shortest path and the route of p is a shortest path, then the trace of* Switch(τ) *is a shortest path.*

The definition of time path sequence remains unchanged: a sequence $\tau_0, \tau_1 \ldots$ is a *time path sequence* if τ_0 is a time path, and $\tau_{j+1} = Switch(\tau_j)$ for all $j \geq 0$.

The lemma below is analogous to Lemma 3.7. The proof is omitted.

LEMMA 5.5. *Let $\tau_0, \tau_1, \ldots$ be a time path sequence. Suppose that the rank of τ_i is r, and assume that the rank of τ_j is also r, where $i < j$. Then there exists $i < l < j$ such that the rank of τ_l is strictly smaller than r.*

The lemma below is analogous to Lemma 3.8. We provide a slightly different proof here.

LEMMA 5.6. *Given a schedule of a finite set of finite routes, all time path sequences are finite.*

Proof. Unlike the synchronous case, here the number of possible time paths is not bounded. However, the number of ranks is bounded by $2\Sigma_{i=1}^{k} d_i$. Assume, for contradiction, that there exists an infinite time path sequence. Consider the corresponding infinite sequence of ranks. Let r be the smallest rank that appears infinitely often in the rank sequence. By Lemma 5.5 there must exist a rank $r' < r$ that appears infinitely often as well, a contradiction to the minimality of r. □

The theorem below, obviously, is the analog of Theorem 3.1. The proof of Theorem 3.1 is a proof for Theorem 5.7 as is. We restate it here for the sake of completeness.

THEOREM 5.7. *Let $p_1, \ldots, p_k$ be a set of packets whose routes are $R_1, \ldots, R_k$, respectively. Suppose each R_i is a shortest path of length d_i, $1 \leq i \leq k$. Then in any greedy asynchronous schedule each packet p_i arrives* *at its destination within $d_i + \lfloor (k-1)/w \rfloor$ time units, where w is the width of the links.*

6. Discussion

In this paper we studied the performance of greedy queuing policies. Our main result shows that any greedy policy guarantees a certain pipelining property, when the paths traversed by the packets are shortest paths. We stress that the correct interpretation of this result is not a particular protocol. Rather, it proves that in the case of unbounded queues at the nodes, greedy policies cannot be "catastrophic."

The assumption that the packets traverse shortest paths is crucial for our proof. It would be interesting to extend the results beyond the shortest paths assumption. A natural candidate would be a variant of "nearly shortest paths." Unfortunately, in all the formulations that we considered, the bounds that we achieved were not interesting.

Our result is tight in the sense that there are scenarios in which the delivery time is exactly $d + \lfloor (k-1)/w \rfloor$. However, since this bound does not assume any other properties of the routing paths, it is conceivable that a better bound can be obtained when additional information is assumed. For example, suppose that all packets start at the same time. Then, for a packet which is delayed, the configuration of the paths implies a smaller number of delays in subsequent steps. This property may be captured better by a potential function. Finding such a potential function seems to be an interesting open problem.

Another interesting open problem is to find the corresponding bound for the case of bounded queues, i.e., when a packet may be delayed at a node due to lack of space in the node at the other endpoint.

References

1. D. Bertsekas and R. Gallager, "Data Networks," Prentice-Hall, Englewood Cliffs, NJ, 1987.
2. I. Cidon, S. Kutten, Y. Mansour, and D. Peleg, Greedy packet scheduling, *in* "Proceedings, 4th International Workshop on Distributed Algorithms, Bari, Italy, September 1990."
3. T. Leighton, B. Maggs, and S. Rao, Universal packet routing algorithm, *in* "29th Annual Symposium on Foundations of Computer Science, White Plains, NY, October 1988," pp. 256–269.
4. J. Martin, "SNA: IBM's Networking Solution," Prentice-Hall, Englewood Cliffs, NJ, 1982.

5. J. McQuillan, I. Richter, and E. Rosen, The new routing algorithm for the ARPANET, *IEEE Trans. Commun.* **28**, No. 5 (1980), 711–719.
6. P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe, Scheduling data redistribution in distributed databases, *in* "Proceedings 6th Intl. Conf. on Data Engineering, Los Angles, CA, Feb. 1990," pp. 166–173; also, Technical Report UF-CIS-TR-90-7, Computer and Information Sciences Department, University of Florida.
7. A. Tannenbaum, "Computer Networks," Prentice–Hall, Englewood Cliffs, NJ, 1981.
8. L. G. Valiant and G. J. Brebner, Universal schemes for parallel communication, *in* "Proceedings, 13th Annual ACM Symposium on Theory of Computing, Milwaukee, WI, May 1981," pp. 263–277.