Greedy Packet Scheduling on Shortest Paths

PRELIMINARY VERSION

Yishay Mansour* Boaz Patt-Shamir†

Abstract

We investigate the simple class of greedy scheduling algorithms, that is, algorithms that always forward a packet if they can. Assuming 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 a 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. Furthermore, the result holds in the asynchronous model, using the same proof technique.

The generality of our result is demonstrated by a variety of 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.

*Aiken computation Laboratory, Harvard University, Cambridge, MA 02138. Supported in part by ONR under contract N00014-85-K-0445.

†Laboratory for Computer Science, MIT, Cambridge, MA 02139. Supported in part by DARPA under Contract N00014-87-K-0825.

1 Introduction

The performance of a routing protocol is evaluated by various measures, such as global network throughput, maximum packet delivery time, storage requirement etc. Conceptually, we may break the task of routing a set of packets across communication network into two subproblems, which we refer to as *path selection* and *queue policy*. The path selection determines the routes that packets traverse, while the queue policy determines which of the packets 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 transmission paths. The selection may be made at the destination, as in circuit switching networks [Mar82], or while the packet progresses in the network, as is common in packet switching networks [MRR80].

The need for a queue policy emerges from the following physical restriction of the network. The capacity of the links is bounded, i.e., only one packet can progress over a link at a time. However, it is conceivable that more than one packet wishes to traverse the same link simultaneously. The task of a queue policy is to schedule the pending packets over the link. In practice, queue policies exhibit assorted combinations of FIFO, fixed priorities, some sort of congestion control etc. (see [BG87, Tan81]). In most cases the queue policy is *greedy*, that is, a

packet is delayed only if another packet currently progresses over the link.

The isolation of path selection and queue 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. The abstraction of routing protocols, as presented here, was first studied by Leighton, Maggs and Rao [LMR88]. 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. [LMR88] 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. [CKMP90] conjectured that in the synchronous model, if the path selection algorithm produces shortest paths, then any greedy queue policy delivers all k packets within $d+k-1$ time units. Previously, only special cases of this general conjecture were proved. In their classical work on randomized routing on a hypercube, Valiant and Brebner [VB81] 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 queue policy, denoted by Min-Went (the queue policy at the nodes is to first forward packets that traversed the least distance so far).

In [CKMP90] it is also shown that if 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.

Recently, and independently, Rivera-Vega et al. [RVVN91] proved that scheduling packets on shortest paths, using fixed priority to resolve conflicts requires no more than $d+k-1$ time units.

In this work we settle the conjecture posed in [CKMP90] in the affirmative. Specifically, we consider the general scenario in which k packets traverse shortest paths, and the queue policy is *any* greedy queue policy. We show that any such schedule is guaranteed to deliver packet p_i in no more than d_i+k-1 time units, where d_i is the number of links p_i traverses. 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 any time. This generality enables us to derive a few interesting applications.

1. We derive a simple protocol for the case where the route lengths are distinct and the starting times of all packets are identical. This protocol is guaranteed to deliver all packets in $\max\{d_i\}$ time units, which is clearly optimal.[1]

2. In the *dynamic* model, the total number of packets is not bounded, and packets are contin-

[1] A similar protocol appears in [RVVN91].

uously generated and delivered. We show that if one selects a greedy queue 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 $d_i + \ell$ time units, where ℓ is the number of packets in the network when p_i is generated.

3. 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 queue policy, even in the asynchronous model, that every $O(n)$ time units n 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.

For simplicity of presentation, we prove our results in the synchronous setting, and discuss the asynchronous case in the appendix. 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, using fixed priority queue policy, are presented in Section 4. In Section 5 we discuss the throughput of a dynamic network. In Appendix A we describe the asynchronous model and sketch how do the results carry through.

2 The synchronous model

In this section we define formally 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*, $R_i = (v_0^i, \ldots, v_{d_i}^i)$, $1 \leq i \leq k$. 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 $1 \leq i \leq k$.

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 any time step. 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 $1 \leq i \leq k$ and time steps $t_s^i \leq t < t_e^i$, either packet p_i *does not make 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 capacity condition*. Intuitively, this condition is that the schedule may not send two packets over the same link at a single time step. Formally, a schedule satisfies the link capacity condition if whenever packets p_i and p_j meet at times t and $t+1$, then both p_i and p_j do not progress at time t.

Packet p_i is said to *delay* packet p_j in 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 packets waiting to be forwarded on some link, then one of these packets is forwarded. Formally, if p_j does not progress at time t_0, then there exists a packet p_i that delays p_j at time t_0.

In Appendix A we generalize the definitions of this section 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 $p_1, \ldots, p_k$ be a set of packets whose routes are $R_1, \ldots, R_k$. 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 $d_i + k - 1$ time units.

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 (fixed) 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 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 formally defined 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 path induced by $L(\tau, t_s), \ldots, L(\tau, t_e)$ is the *trace* of the τ. 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. This, in conjunction with the fact that the final time path has the same length the original route has, proves that the time for this packet is bounded by $d_i + k - 1$.

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 the packet that delayed packet $\tau(t)$ at time t. The existence of such a packet p 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 $t' > t$. 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 all $t_s \leq t \leq t_e$, $Switch(\tau)(t)$ is defined as follows.

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

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'll 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.

We first show that applying *Switch* to a time path yields a time path.

Lemma 3.4 Let τ be a time path with duration $[t_s, t_e]$, and assume that *Switch* is applicable to τ. Then $Switch(\tau)$ is a time path with duration $[t_s, t_e]$.

Proof: Let $\tau' = Switch(\tau)$. 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 $t = t_1$. ∎

The following three lemmas are immediate from the definition of *Switch*.

Lemma 3.5 Assume that *Switch* is applicable to time path τ, and let t_0 be the rank of τ. Then $Switch(\tau)$ is not delayed at time t_0.

Proof: Let $\tau' = Switch(\tau)$ and 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 Lemma 3.4, 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. ∎

Lemma 3.6 Let τ be a time path of duration $[t_s, t_e]$, and assume that *Switch* is applicable to τ. Let $\tau' = Switch(\tau)$. Then $L(\tau', t_s) = L(\tau, t_s)$, and $L(\tau', t_e) = L(\tau, t_e)$.

Proof: Consider the destination first. By the definition of *Switch*, $\tau'(t_e) = \tau(t_e)$, 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 $L(\tau', t_s) = S(p, t_s) = L(\tau, t_s)$. ∎

Lemma 3.7 Let τ be a time path of duration $[t_s, t_e]$, and 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: Immediate from the definition of *Switch* applicability. ∎

Next, we show that *Switch* preserves the shortest path property. This follows from the fact that both paths are shortest paths.

Lemma 3.8 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 $Switch(\tau)$ 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 are shortest paths between their endpoints, and since they have the same endpoints, their lengths are equal. Therefore, the length of the trace of $Switch(\tau)$ is equal to the length of the trace of τ. Having this fact, together with the fact that the traces of τ and $Switch(\tau)$ share the same endpoints (Lemma 3.6), and with the assumption that the trace of τ is a shortest path, we conclude that the trace induced by $Switch(\tau)$ 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* to a time path.

Definition 3.9 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 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 3.10 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: Assume, by the way of contradiction, that the rank of τ_l, $i < l < j$, is at least r. By Lemma 3.5, τ_{i+1} is not delayed at time r. By the definition of *Switch*, the rank of a time path τ must be equal to some time step in which τ is delayed. Hence the rank of τ_l must be strictly greater than r for all $i < l < j$. This implies that for all $i < l < l' < j$, $\tau_l(t) = \tau_{l'}(t)$ 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 3.11 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^D, where $D = kd$, where k is a bound on the number of packets and d is a bound on the length of the routes). Suppose now, by the way of contradiction, that there exists an infinite time path sequence. Then there must exist a cyclic time path sequence, i.e., time paths $\tau_j, \tau_{j+1}, \ldots, \tau_{j+c}$ 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 3.10, 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 $p_1, \ldots, p_k$ be a set of packets whose routes are $R_1, \ldots, R_k$. 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 $d_i + k - 1$ time units.

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 $\tau_0, \tau_1, \ldots$ 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 3.11, 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. Hence, adding the fact that p_i does not delay τ^* (by Lemma 3.7), we have that the total number of delays in τ^* is at most $k-1$, which implies that the duration of τ^* is at most its trace length plus $k-1$. From Lemma 3.8 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 $d_i + k - 1$ time steps. ∎

As a final remark we stress the fact 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 with fixed priorities

Theorem 3.1 provides us with an interesting insight into the relationship between queue policies and the path selection schemes. In this section we give a few simple applications of the result. We employ the *priority* queue policy, in which scheduling conflicts are resolved by a global fixed priority ordering. Specifically, each packet is assigned a priority, and if a conflict arises, then the packet with the higher priority wins (ties may be broken arbitrarily). We remark that the priorities for the queue policies presented in this section are local, in the sense that a packet can compute its priority independently of the other packets.

Formally, the priority is a mapping $\mathcal{H}$ of packets to integers. The priority queue policy is the rule that packet p delays p' only if $\mathcal{H}(p) > \mathcal{H}(p')$. We call the set $E_p = \{p' : \mathcal{H}(p') \geq \mathcal{H}(p)\}$ the set of *effective packets* with respect to p, and denote $k_p = |E_p|$.

Fix a priority queue 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 time for the delivery of p, we may assign $k = k_p$.

4.1 Distinct path lengths

We begin with the case where all route lengths are distinct and all packets start at the same time. For the queue policy in which the packet with the longer path has the higher priority, we obtain the following optimal result.

Theorem 4.1 Let $R_0, \ldots, R_m$ be a collection of routes with lengths $d_0, \ldots, d_m$, respectively. Assume that all routes are shortest. If the lengths of the routes are distinct, then there is a queue policy that ensures that all packets are delivered within $\max_i \{d_i\}$ time units.

Proof: Define a priority queue policy by $\mathcal{H}(p_i) = d_i$. Without loss of generality, assume that $d_i > d_{i+1}$ for $0 \leq i < m$. Note that $d_i \geq d_0 + i$. Since packet p_i can be delayed only by $p_0, \ldots, p_{i-1}$, applying Theorem 3.1 with $k_{p_i} = i + 1$ shows that packet p_i arrives within $d_i + i \leq d_0 = \max_i \{d_i\}$ time units. ∎

4.2 Dynamic model

In the *dynamic model*, we consider a network in which packets are continuously generated. Denote the time packet p is generated (and delivered, respectively) by t_s^p (resp., t_e^p). The set of *packets in transit* at time t is defined as $M_t = \{p : t_s^p \leq t < t_e^p\}$. Denote by ℓ_p the number of packets in transit when p is sent, i.e., $\ell_p = |M_{t_s^p}|$. Defining a queue policy in which a packet p can be delayed only by packets in transit when p is generated, we obtain the following result.

Theorem 4.2 In a dynamic network, if the packets traverse shortest paths, then there is a queue policy

that delivers any packet p within $d_p + \ell_p$ time units, where d_p is the length of the route of p.

Proof: Define $\mathcal{H}(p) = -t_s^p$. ∎

5 Throughput of a network

In this section we consider the dynamic setting, as described in Section 4.2. 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 acknowledgement is sent back to the source. A packet is sent only after an acknowledgement for the previous packet is received. We assume that the packets and the acknowledgements traverse shortest paths. Note that this protocol guarantees that in any point at time, for each node there is at most one message (either a packet or an acknowledgment) in transit.

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 n packets are delivered, as long as the queue policy is greedy.

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

Lemma 5.1 In any $3n$ consecutive time units, at least n messages (either regular packets or acknowledgements) are delivered.

Proof: As mentioned above, there are always n messages in transit in the network. Let t be any point in time, and let $m_1, \ldots, m_n$ be the the messages in transit at time t. If all 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 $2n$ messages in transit in this time interval, and therefore, at least n messages are delivered (again, since in every point in time there are n 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 5.2 The generic protocol, coupled with any greedy queue policy, in any $\Theta(n)$ consecutive time units, at least n packets are delivered.

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

Consider a time interval of length $9n$. By Lemma 5.1, at least $3n$ messages are delivered in any such time interval. If less than n *packets* are delivered, then, by the pigeonhole argument, there must exist a node for which the number of received acknowledgements is greater than the number of delivered packets by at least two, a contradiction. ∎

Theorem 5.2 applies to any greedy queue policy when coupled with the above congestion control protocol. We remark that for the FIFO queue policy, a packet is delivered within $O(n^2)$ time units in the worst case. Theorem 5.2 shows that the throughput is much better: every $O(n)$ time units n packets are delivered (this may be viewed as an $O(n)$ average-case delivery time).

References

[BG87] Dimitri Bertsekas and Robert Gallager. *Data Networks.* Prentice Hall, Englewood Cliffs, New Jersey, 1987.

[CKMP90] Israel Cidon, Shay Kutten, Yishay Mansour, and David Peleg. Greedy packet scheduling. In *Proc. 4th International Workshop on Distributed Algorithms*, Bari, Italy, September 1990.

[LMR88] Tom Leighton, Bruce Maggs, and Satish Rao. Universal packet routing algorithm. In *29th Annual Symposium on Foundations of Computer Science*, pages 256–269, White Plains, NY, October 1988.

[Mar82] James Martin. *SNA: IBM's Networking Solution.* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[MRR80] John McQuillan, Ira Richer, and Eric Rosen. The new routing algorithm for the ARPANET. *IEEE Trans. Comm.*, 28(5):711–719, May 1980.

[RVVN91] Pedro I. Rivera-Vega, Ravi Varadarajan, and Shamkant B. Navathe. The file redistribution scheduling problem. manuscript, 1991.

[Tan81] Andrew Tannenbaum. *Computer Networks.* Prentice Hall, 1981.

[VB81] Leslie G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, Milwaukee, WI, May 1981.

A Schedules in the asynchronous setting

In this appendix we discuss why the proof of Theorem 3.1 applies to the asynchronous model. The statement of the problem in this model differs from the synchronous model only in the following point: every packet sent over a link arrives in arbitrary time. Formally, assume that the route of packet p_i is $v^i_0, \ldots, v^i_{d_i}$. We associate with each packet p_i the time in which the *send events* $s^i_0, \ldots, s^i_{d_i-1}$ and the *receive events* $r^i_1, \ldots, r^i_{d_i}$ occur. Time s^i_j is the time in which packet p_i is sent from node v^i_j, and r^i_{j+1} is the time in which p_i is received in the other end of the link, i.e., node v^i_{j+1}. For any schedule, the time of the events must satisfy $r^i_j \leq s^i_j < r^i_{j+1}$ for all $1 \leq i \leq k$ and $0 \leq j < d_i$. For the purpose of time analysis, we assume that the a packet progresses over a link at most one time unit, i.e., $0 < r^i_{j+1} - s^i_j \leq 1$ for $1 \leq i \leq k$ and $0 \leq j < d_i$.

Recall that the link capacity condition is that at any point in time, there is at most one packet progresses the link. Formally, if $(v^i_j, v^i_{j+1}) = (v^{i'}_{j'}, v^{i'}_{j'+1})$ for some j, j' and $i \neq i'$, then either $s^i_j \geq r^{i'}_{j'+1}$ or $s^{i'}_{j'} \geq r^i_{j+1}$.

A key difficulty in adapting the proof from Section 3 to 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.

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 $S(p_i, t) = v^i_j$, where j is such that $r^i_j \leq t < r^i_{j+1}$ 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.

The definition of *meet* remains unchanged.

Packet p_i is said to *progress* at the time intervals $[s_j^i, r_{j+1}^i)$ for $0 \leq j \leq d_i$.

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 $(v_j^i, v_{j+1}^i) = (v_{j'}^{i'}, v_{j'+1}^{i'})$, $t_0 = \max\left\{r_{j'}^{i'}, s_j^i\right\}$, $t_1 = \min\left\{r_{j+1}^i, s_{j'}^{i'}\right\}$ and $t_0 < t_1$.

A queue policy is *greedy* if a packet p_i does not progress only if there exists a packet p_j that delays it.

We now turn to formalize the definition of time path in the continuous fashion.

Definition A.1 Let $t_s \leq t_e$ 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 $t_s < t_0 \leq t_e$

$$\lim_{\substack{t \to t_0 \\ t < t_0}} S(\tau(t), t_0) = S(\tau(t_0), t_0) \ .$$

The definitions of *location*, *trace* and *duration* of a time path remain the same.

The definition of *Switch* requires no change under the revised definitions of a time path and the notion of a packet delaying another. 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 $2\sum_i d_i$ possible ranks.

In what follows, we restate the lemmas and discuss how their proofs can be transformed to apply to the asynchronous model.

The lemma below is analogous to Lemma 3.4.

Lemma A.2 Let τ be a time path with duration $[t_s, t_e]$, and assume that *Switch* is applicable to τ. Then *Switch*(τ) is a time path with duration $[t_s, t_e]$.

Proof: Let $\tau' = Switch(\tau)$. Obviously, τ' is defined in the time interval $[t_s, t_e]$. Let $[t_0, t_1)$ be the detour time, and let p be the detour packet. Clearly, $\lim_{\substack{t \to t_0 \\ t < t_0}} S(\tau(t), t_0) = L(\tau, t_0)$ for all time steps $t_s < t \leq t_e$ satisfying $t \neq t_0$ and $t \neq t_1$. Suppose now that $\lim_{\substack{t \to t_0 \\ t < t_0}} \tau(t) = p_0$. Since p delays p_0 at time t_0, we have that $S(p_0, t_0) = S(p, t_0)$, which implies $\lim_{\substack{t \to t_0 \\ t < t_0}} S(\tau'(t), t_0) = L(\tau', t_0)$. Similarly, the condition holds for $t = t_1$. ∎

The lemma below is analogous to Lemma 3.5.

Lemma A.3 Assume that *Switch* is applicable to time path τ, and let t_0 be the rank of τ. Then *Switch*(τ) is not delayed at time t_0.

For Lemma A.3, we notice that if τ is delayed at time t_0, then it is delayed by some packet p at time t_0 through t', and we substitute t' for $t_0 + 1$ in the proof of Lemma 3.5.

The lemma below is analogous to Lemma 3.6.

Lemma A.4 Let τ be a time path of duration $[t_s, t_e]$. Assume that *Switch* is applicable to τ, and let $\tau' = Switch(\tau)$. Then $L(\tau', t_s) = L(\tau, t_s)$ and $L(\tau', t_e) = L(\tau, t_e)$.

The lemma below is analogous to Lemma 3.7.

Lemma A.5 Let τ be a time path of duration $[t_s, t_e]$, and suppose that p and $\tau(t_e)$ meet at time t_e. If p delays τ at time $t \in [t_s, t_e)$ then *Switch* is applicable to τ.

The lemma below is analogous to Lemma 3.8.

Lemma A.6 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.

The lemma below is analogous to Lemma 3.10.

Lemma A.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.

The lemma below is analogous to Lemma 3.11. We provide an alternative proof here.

Lemma A.8 Given a schedule of a finite set of finite routes, all time path sequences are finite.

Sketch of proof: Unlike the synchronous case, here the number of possible time paths is not bounded. However, the number of ranks is bounded by $2\sum_{i=1}^{k} d_i$. Assume, by the way of 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 A.7 there must exist a rank $r' < r$ that appears infinitely often as well, a contradiction. ∎

The theorem below, obviously, is the analog of Theorem 3.1.

Theorem A.9 Let $p_1, \ldots, p_k$ be a set of packets whose routes are $R_1, \ldots, R_k$. 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 $d_i + k - 1$ time units.

The proof of Theorem 3.1 is a proof for Theorem A.9 as is.