Scheduling Unsplittable Flows Using Parallel Switches

Saad Mneimneh, Kai-Yeung Siu
Massachusetts Institute of Technology
77 Massachusetts Avenue Room 1-107,
Cambridge, MA 02139

***Abstract*— We address the problem of scheduling unsplittable flows using a number of switches in parallel. This has applications in optical switching and eliminates the need for re-sequencing in traditional packet switching. The use of parallel switches is becoming increasingly popular [1], [2] since it provides a way of building a high-speed switch while overcoming the speedup requirement imposed on the switch. Unlike packet switching however, we will assume that flows cannot be split across switches. This constraint adds a new dimention to the problem: various questions such as obtaining the best schedule, i.e. the schedule with the maximum throughput possible, become NP hard.**

Our problem is a special case of the general unsplittable flow problem, where in a directed capacitated graph containing a number of commodities with demands, the goal is to obtain a flow that does not violate capacity and in which all demands are satisfied and every commodity flows along a single path. In this paper, we are not going to address the general problem. Rather, we will study the special case of scheduling unsplittable flows using parallel switches, and present some simple approximation algorithms to various aspects of the problem with no speedup. We also define a speedup version of the problem and discuss under what speedup we can fully schedule an admissible set of flows.

***Keywords* — Unsplittable flows, optical switching, re-sequencing, approximation algorithms, speedup.**

I. Introduction

THE unsplittable flow problem is studied well in the literature (see [3] for good references). In this paper, we study a special case of the unsplittable flow problem, namely, the problem of scheduling unsplittable flows using a number of switches in parallel. Before we proceed to the description of the architecture and the problem statement, we first motivate our approach and list a number of assumptions. Three main concerns motivated our decision for not to split the flows:

- Per-flow guarantees: We would be able to achieve per-flow guarantees since each flow will have a dedicated path and bandwidth.
- Re-sequencing: Since packets cannot be out-of-order at the output port anymore, we will eliminate the need for re-sequencing, which is the main drawback of the architecture described in [2].
- Optical flows: We would accomodate for optical routers since optical flows are naturally unsplittable.

In the ideal situation, we would like our scheduling algorithm to be an *online* algorithm. We also would like it to be *oblivious* in the sense that it would be able to schedule the flows without any knowledge of the remaining capacities on the links connecting the switches to the output ports, since this knowledge is probably hard to obtain without any communication between the input and output ports or between the input ports themselves. We will prove that with no speedup (which will be defined later), a fairly general notion of an online algorithm, that we call *greedy*, cannot schedule a subset of the flows in a way to obtain a throughput that is a positive fraction of the maximum throughput possible, even if the flows are admissible. Throughout the paper, we will assume that the algorithm is not *oblivious* and has exact knowledge of the remaining capacities on the links connecting the switches to the output ports. In the future though, we would like to investigate how to relax this knowledge requirement.

The switching architecture that we are going to use is depicted in Fig 1. Each input and output is connected to all k switches.

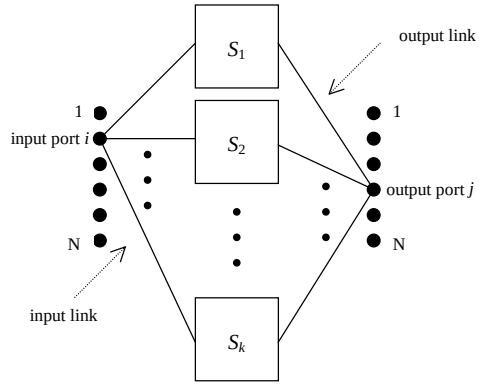

Fig. 1. The parallel switch architecture

We assume without loss of generality that the number of inputs and the number of outputs are equal to N. Each link has capacity 1 and, as a consequence, each switch will be able to handle N units of bandwidth, where N is the number of input and output ports. Each flow is at most 1 unit of bandwidth. In the speedup version of the problem, each link has capacity S (where $S > 1$), and therefore, each switch handles SN units of badndwidth. We choose not to affect the demand (individual flow size) with speedup and therefore, each flow will still be at most 1 unit of bandwidth in the speedup version of the problem. Therefore, given a

Emails: saad@mit.edu, siu@perth.mit.edu. This research is done at the Massachusetts Institute of Technology and is supported by the Networking Research Program of the National Science Foundation, NSF Award 9973015.

0-7803-7400-2/02/$17.00 © 2002 IEEE

set of flows each of which is at most 1, we would like to unsplittably schedule the flows, which means to assign flows to the switches such that all link capacities are not exceeded i.e. each link will handle at most 1 unit of bandwidth (or S units of bandwidth in the speedup version). A set of flows is *admissible* iff the sum of flows at every input and every output is at most k. We are going to address the following questions which are generally addressed in the literature:

- Maximization: Given a set of flows, can we unsplittably schedule a subset whose throughput is the maximum throughput possible?
- Number of rounds: Given a set of flows, what is the minimum number of rounds that are needed to unsplittably schedule all the flows?
- Speedup: Given an admissible set of flows, what is the minimum speedup needed to unsplittably schedule all the flows in one round? This is known as the congestion factor in the literature [3].

Note that all the problems stated above are NP-hard. With regard to the maximization problem, we present some simple approximations algorithms that guarantee a constant fraction of the maximum throughput possible. As for the number of rounds, Du et al. shows in [5] that $\lceil \frac{17k-5}{6} \rceil$ switches with no speedup are enough to unsplittably schedule an admissible set of flows. Therefore, 3 rounds are enough to unsplittably schedule an admissible set of flows. We will provide a 4-approximation algorithm for the number of rounds when the set of flows is not admissible. Du et al. shows also in [5] that it is possible to unsplittably schedule an admissible set of flows with a speedup $S > 1 + \frac{k-1}{k}B$, where k is the number of switches and B is an upper bound on the size of any flow. Therefore, in our case, a speedup of 2 will be enough. We will address the same speedup question for the case of online algorithms.

The rest of the paper is organized as follows. Section II establishes a theoretical framework that we are going to use for answering the various questions above. Section III addresses the approximability of the maximization problem, Section IV addresses the number of rounds needed, and Section V addresses the speedup requirement.

II. Theoretical Framework

In this section, we define a term that we call *blocking factor*. Before we go to the main definition, we need some preliminary definitions. Assume that we have a set of flows F and that we schedule a subset of flows $G \subseteq F$. In other words, G is the set of flows in F that are passing through the parellel switch architecture. Let $in_{s,f}$ be the input link connecting the input port of flow f to switch s. Similarly, let $out_{s,f}$ be the output link connecting the output port of flow f to switch s. Let $u(e)$ be the amount of flow on link e. An input link e in switch s is *blocking* if there exists a flow $f \notin G$ such that $e = in_{s,f}$ and $u(e) \geq u(out_{s,f})$. Similarly, an output link e in switch s is *blocking* if there exists a flow $f \notin G$ such that $e = out_{s,f}$ and $u(e) \geq u(in_{s,f})$.

Definition 1 (blocking factor) The blocking factor β is defined as follows:

$$\beta = \begin{cases} \min_{e \in B} u(e) & \text{if } B \neq \emptyset \\ \infty & \text{if } B = \emptyset \end{cases}$$

where B is the set of all blocking links.

Note that for any switch s and every flow $f \notin G$, at least one of $in_{s,f}$ and $out_{s,f}$ is a blocking link, and hence f has to use at least one blocking link in s in order to be routed through s. Therefore, every flow $f \notin G$ has to use at least one blocking link in order to be routed through the parallel switch architecture.

The blocking factor is a measure of how large the throughput of G is, since for every flow $f \notin G$, we look at how much flow in G is going through the switches, either from the input port of f or to the output port of f. A large blocking factor is therefore an indication of a high utilization of the parallel switch architecture and hence of a high approximability of the maximization problem.

III. Maximization

In this section, we establish a loose connection between the blocking factor and the approximability of the maximization problem in scheduling unsplittable flows. More precisely, we show that a blocking factor β implies a $\frac{\beta}{2+\beta}$-approximation. We first prove that a fairly general notion of an online algorithm, that we call *greedy*, cannot achieve any positive approximation factor.

Definition 2 (greedy) A *greedy* scheduling algorithm is an online algorithm that satisfies the following three conditions:
- When a flow arrives, the algorithm has to decide immediately whether to accept or reject the flow.
- If a flow can be scheduled without violating any link capacity, then the algorithm has to accept the flow and assign it to one of the parallel switches.
- once a flow is accepted and assigned to a switch, it cannot be re-routed.

It is worth mentioning that Tsai et. al. proved in [4] that a multirate clos network is wide-sense non-blocking only if the number of switches is at least $3k-2$. Since a multirate clos network is a special case of our architecture, the result still applies. This is equivalent to the statement that any *greedy* algorithm requires at least $3k-2$ switches in order to unsplittably schedule all flows.

Theorem 1: For any $0 < \epsilon \leq 1$, there is no *greedy* ϵ-approximation algorithm for the problem of maximizing the throughput in scheduling unsplittable flows, even with an admissible set of flows.

Proof: The proof is by construction of a particular instance of the problem where, for a given ϵ, $N = (\lfloor \frac{1}{\epsilon} \rfloor + 1)k$. We divide the input ports into two sets I_1 and I_2, where I_1 contains the first k input ports. Similarly, we divide the output ports into two sets O_1 and O_2, where O_1 contains

the first k output ports. Assume that for every input port $i \in I_1$, we schedule an amount of flow equal to 1 from input port i to output port i in all k switches, except for switch i where we schedule a flow of size $1-\epsilon'$ from input port i to output port i, where $0 < \epsilon' < \frac{1}{2}$. Given this setting, any future flow for output $j \in O_1$ has to be assigned to switch j, since all links to output j in all other switches are fully utilized. We will later describe how we can force such a setting. Next, from every input port $i \in I_2$, we receive k flows of size ϵ'' for all the output ports in O_1. Note that for the admissibility condition to hold at an output port $j \in O_1$, we require that $\lfloor \frac{1}{\epsilon} \rfloor k\epsilon'' \leq \epsilon'$; we can assume equality. The *greedy* algorithm will have to schedule these flows and, as argued above, will assign flows for output j to switch j. For all $i \in I_2$, this will make all k links from input port i partially utilized. Next, for each $i \in I_2$, we receive $k-1$ flows of size 1 and one flow of size $1-k\epsilon''$, all from input port i to output port i. Note that the admissibility condition at input ports $i \in I_2$ still holds since $\epsilon'' \times (k) + 1 \times (k-1) + 1 - k\epsilon'' = k$. For each $i \in I_2$, the *greedy* algorithm can only schedule the last flow of size $1-k\epsilon''$ from input port i to output port i, since every link from input port i is now partially utilized. So far, the sum of all flows is $kN - k\epsilon'$. So we can still have $k\epsilon'$ additional amount of flow (ϵ' from each input port $i \in I_1$), and without loss of generality, we can assume that the *greedy* algorithm is able to schedule them. Note that the maximum throughput possible is at least $k^2 - k\epsilon' + \lfloor \frac{1}{\epsilon} \rfloor k(k-1)$ which consists of the initial setting of the switches in addition to assigning to each of the first $k-1$ switches a flow of size 1 from input port i to output port i, for all $i \in I_2$. Therefore, the maximum throughput possible is $O((1 + \lfloor \frac{1}{\epsilon} \rfloor)k^2)$. The throughput achieved by the *greedy* algorithm is $k^2 + (1-k\epsilon'')\lfloor \frac{1}{\epsilon} \rfloor k$, which is $O(k^2)$. Therefore, the approximation factor is at most

$$\frac{O(k^2)}{O((1 + \lfloor \frac{1}{\epsilon} \rfloor)k^2)}$$

which goes to $\frac{1}{\lfloor \frac{1}{\epsilon} \rfloor + 1} < \epsilon$ when k grows large enough. Now we illustrate how we can force the setting described at the beginning of this proof, namely, for a given i, scheduling an amount of flow equal to 1 from input port i to output port i in all k switches, except for switch i where we schedule a flow of size $1-\epsilon'$. We begin by receiving flows, all of which are greater than $\frac{1}{2}$, from input port i to output port i while increasing the size of the flow in every time by a small amount. In every time, the *greedy* algorithm will choose a different switch to schedule the flow, since two flows cannot be assigned to the same switch without exceeding the link capacities. After the *greedy* algorithm chooses switch i, which has to happen at some point, we start fully utilizing links from input port i to all the switches that have not been chosen yet by the *greedy* algorithm, by receiving a sufficient number of flows of size 1 from input port i to output port i. After that, among all the switches which can take more flows from input port i to output port i, switch i will have the largest amount of flow going through it from input port i to output port i. We start receiving flows in such a way to cause the *greedy* algorithm to choose switches in the same order it had before, by receiving the flows in decreasing size this time and fully utilizing every link from input port i, except the one going to switch i. For instance, if the amount of flow going from input port i to output port i through switch s is b, we receive a flow of size $1-b$ from input port i to output port i, which has to be assigned to switch s. We stop just after fully utilizing all the links from input port i except the one going to switch i. Hence switch i will have $1-\epsilon'$ amount of flow going from input port i to output port i, where $0 < \epsilon' < \frac{1}{2}$. ■

Note that it is possible to prove Theorem 1 using a simpler instance with two switches. In that case however, the total amount of flow presented to the switches will be less than the total capacity of the two switches. The instance used in the proof above has the special property that the total amount of flow received is $C = kN$, where C is the full capacity of the switches. This implies that the low approximation factor is not due to the abscence of flows, and in other words, adding more flows cannot enhance the approximation factor. So even with enough flow equal to C, for any ϵ, a *greedy* algorithm will not be able to achieve a total throughput greater or equal to ϵC.

We now prove that a blocking factor β implies an approximation factor $\frac{\beta}{2+\beta}$.

Lemma 1: If the blocking factor is β, then the throughput is at least a fraction $\frac{\beta}{2+\beta}$ of the maximum throughput possible.

Proof: If $\beta = \infty$, then there are no blocking links and hence no flows are left out. Therefore, a fraction equal to $\frac{\infty}{\infty+2} = 1$ of the maximum throughput possible is scheduled. If β is finite, then let L be the number of blocking links. Let F be the set of all flows, and $G \subset F$ be the set of flows that are scheduled. Let G' be the set of flows in $F - G$ that are scheduled in the optimal solution. We know that every flow in $F - G$, and hence in G', has to use at least one blocking link in order to be scheduled in one of the switches. This means that the sum of flows in G' cannot be more than L since every blocking link has capacity 1. Moreover, for every blocking link e, $u(e) \geq \beta$ by definition. This means that the sum of flows in G that are passing through blocking links is at least $L\beta/2$, since a flow can pass through at most two blocking links. This implies that the sum of flows in G is at least $\beta/2$ that of G'. The throughput of the optimal solution cannot be more than the sum of flows in $G \cup G'$ by definition of G and G'. Therefore the sum of flows in G is at least $\frac{L\beta/2}{L+L\beta/2} = \frac{\beta}{2+\beta}$ of the maximum throughput possible. ■

Note that the fraction stated in Lemma 1 is also true if we consider the maximum amount of flow going through an ideal switch, where splitting of flows is permissible. Throughout the analysis, we did not rely on the fact that the optimal solution does not allow for flow splitting. The same analysis applies if we compare the throughput to any

other throughput even when splitting occurs.

Using the result above, we can obtain a $\frac{1}{5}$-approximation algorithm for the problem of maximizing the throughput. The algorithm is described below:

Algorithm A:
We divide the flows into two groups: large flows and small flows. Flows that are greater than $\frac{1}{2}$ are considered large, all other flows are considered small. The algorithm starts by scheduling large flows first, in an arbitrary way, until no more large flows can be assigned to the switches. Then it schedules small flows, in an arbitrary way, until no more small flows can be assigned to the switches.

Lemma 2: Algorithm A guarantees a blocking factor $\beta > \frac{1}{2}$.

Proof: To prove that the blocking factor $\beta > \frac{1}{2}$, assume the opposite. This implies that there is a blocking link e in some switch s such that $u(e) \leq \frac{1}{2}$. By the definition of a blocking link, there exists flow f that is left out, such that either $e = in_{f,s}$ or $e = out_{f,s}$. As a consequence, $u(in_{f,s}) \leq \frac{1}{2}$ and $u(out_{f,s}) \leq \frac{1}{2}$. This means that flow f cannot be a small flow, since otherwise, it could have been assigned to switch s before the algorithm had stopped. So f must be a large flow. But since $u(in_{f,s}) \leq \frac{1}{2}$ and $u(out_{f,s}) \leq \frac{1}{2}$, only small flows are passing through $in_{f,s}$ and $out_{f,s}$, which contradicts the way the algorithm favors large flows first. Therefore, $\beta > \frac{1}{2}$. ■

Theorem 2: There exists a $\frac{1}{5}$-approximation algorithm for the problem of maximizing the throughput in scheduling unsplittable flows.

Proof: *Algorithm A* is a polynomial time algorithm. By Lemma 2, *Algorithm A* guarantees a blocking factor of $\beta > \frac{1}{2}$, which by Lemma 1, implies a $\frac{1}{5}$-approximation for the problem of maximizing throughput. ■

In the following section, we will describe a $\frac{1}{3}$-approximation algorithm for the problem of maximizing throughput, when the set of flows is admissible.

IV. Number of Rounds

The fact that it might be unfeasible to schedule all flows unsplittably, even if the admissibility condition holds, motivates the idea of rounds. In this section, we ask how many rounds are needed to unsplittably schedule all the flows. The authors in [3] provide an algorithm that schedules all flows unsplittably in a general graph with a single source in 5 rounds, given that the cut condition holds. They also show that this leads to a 5-approximation algorithm for the problem of minimizing the number of rounds when the cut condition is not satisfied. The cut condition is a general admissibility condition.

In our case, we provide a 4-approximation algorithm to the minimum number of rounds needed to schedule all flows unsplittably.

When the set of flows is admissible, Du et al. proved in [5] that $\lceil \frac{17n-5}{6} \rceil$ switches are sufficient to unsplittably schedule all flows. This implies that 3 rounds are also sufficient since 3 rounds are equivalent to $3k$ switches. For the sake of completeness, we provide a simple polynomial time algorithm that unsplittably schedules an admissible set of flows using $3k$ switches. Note that when the set of flows is admissible, a polynomial time algorithm that schedules all flows in r rounds implies a $\frac{1}{r}$-approximation algorithm for the problem of maximizing throughput, simply by choosing the round with the maximum throughput, which has to be at least $\frac{1}{r}$ of the sum of all flows. As a consequence, we have a $\frac{1}{3}$-approximation for the problem of maximizing throughput when the set of flows is admissible.

We first describe a 4-approximation algorithm to the minimum number of rounds needed to schedule any set of flows:

Algorithm B:
This algorithm consists of a number of rounds. In each round we run *Algorithm A* on the remaining flows. We stop when all flows have been scheduled.

First we prove a simple lemma.

Lemma 3: If a and b are two integers greater than 0 then $\lfloor \frac{a-1}{b} \rfloor + 1 \geq \frac{a}{b}$.

Proof: $a-1$ can be written as $q \times b + r$ where both q and r are non-negative integers and $r < b$. Then $\lfloor \frac{a-1}{b} \rfloor = q$. Finally, $\frac{a}{b} = \frac{q \times b + r + 1}{b} = q + \frac{r+1}{b} \leq q + 1$. ■

Theorem 3: There exists a 4-approximation algorithm for the problem of minimizing the number of rounds in scheduling unsplittable flows.

Proof: Assume that *Algorithm B* stops after r rounds. Let f be a flow that is scheduled in the r^{th} round. Then we know that in the first $r-1$ rounds, flow f could not be scheduled, and as a consequence, it has a blocking link in every switch during all $r-1$ rounds. Since *algorithm B* runs *algorithm A* in every round, then the blocking factor β is greater than $\frac{1}{2}$ as argued in the proof of Lemma 2. This implies that the total amount of flow coming from the input port of f or going to the output port of f is more than $\frac{1}{2} \times k \times (r-1) = (\frac{r-1}{4}) \times 2k$ during the first $r-1$ rounds. In one round however, we cannot schedule more than $2k$ amount of flow for any pair of input and output ports. This means that we cannot optimally have less than $\lfloor \frac{r-1}{4} \rfloor + 1$ rounds to schedule all the flows including flow f. By Lemma 3, this is at least $\frac{r}{4}$. Therefore, we have the result since *Algorithm B* is a polynomial time algorithm. ■

Next, we present an algorithm that unsplittably schedules an admissible set of flows in 3 rounds. As in the previous algorithms, this algorithm relies on the idea of dividing the flows into two groups.

Algorithm C:
We divide the flows into two groups: large flows and small flows. Flows that are greater than $\frac{1}{3}$ are considered large, all other flows are considered small. The algorithm starts by scheduling large flows first using the rearrangeability property of a clos network (Slepian-Duguid theorem [6]): Since at most $3k-1$ large flows can exist at any port (and each is at most 1), we can unsplittably schedule the large flows using at most $3k-1$ switches, or alternatively $3k$ switches. Then the small flows are scheduled in an arbitrary way. The $3k$ switches correspond to the 3 rounds.

Lemma 4: Let F be a set of flows. If no flow $f \in F$ can be scheduled and each flow $f \in F$ is at most B, then the blocking factor β satisfies $\beta > S - B$, where S is the speedup.

Proof: Assume the opposite. By definition of the blocking factor, there exists a flow $f \in F$ and a switch s such that $u(in_{f,s}) \leq S - B$ and $u(out_{f,s}) \leq S - B$. Therefore, we can assign f to switch s without violating any link capacity (recall that any flow in F, in particular flow f, is at most B). This is clearly a contradiction since flow f cannot be scheduled. Therefore, the blocking facor β satisfies $\beta > S - B$. ■

Theorem 4: Algorithm C unsplittably schedules any admissible set of flows in at most 3 rounds.

Proof: It is enough to show that with *Algorithm C*, no small flows can be left out. To prove this fact, let F be the set of small flows that cannot be scheduled with *Algorithm C*. Applying Lemma 4 to F, $B = \frac{1}{3}$, and $S = 1$, we obtain that the blocking factor β satisfies $\beta > \frac{2}{3}$ in all 3 rounds. Consider a flow $f \in F$. Since f has a blocking link in every switch in all 3 rounds, and the blocking factor is more than $\frac{2}{3}$, the amount of flow coming from the input port of f or going to the output port of f is more than $\frac{2}{3} \times k \times 3 = 2k$. From the admissibility condition however, we know that at most $2k$ amount of flow can exist for any 2 ports. This is a contradiction. Therefore, the set F has to be empty. ■

Corollary 1: There exists a $\frac{1}{3}$-approximation algorithm for the problem of maximizing the throughput in scheduling unsplittable flows when the set of flows is admissible.

Proof: By Theorem 4, *Algorithm C* schedules an admissible set of flows in 3 rounds. The Corollary is true since *Algorithm C* is a polynomial time algorithm and the round with the maximum throughput among all rounds has to have at least $\frac{1}{3}$ of the total amount of flows in the admissible set. ■

In comparision with the work in [3], a $\frac{1}{4.43}$-approximation algorithm is obtained for the problem of maximizing throughput in a general graph with a single source, when the cut condition is satisfied.

Theorem 4 also implies that a speedup of 3 is sufficient to unsplittably schedule an admissible set of flows. This can be achieved by superposing all 3 rounds together to get the effect of one round where each link has capacity 3. In the following section, we prove a stronger result, namely that any *greedy* algorithm can unsplittably schedule an admissible set of flows with a speedup of 3.

V. Speedup

As mentioned before, Du et al. proved in [5] that a speedup of 2 is enough to schedule an admissible set of flows. Note that 2 is also a lower bound on the speedup required to unsplittably schedule an admissible set of flows. To see this, consider $k+1$ flows from input port i to output port j, each of size $\frac{k}{k+1}$. Since there are k switched only, at least 2 flows must be assigned to the same switch. Therefore, the minimum speedup required is $\frac{2k}{k+1}$. For any ϵ, by choosing a large enough k, we can make $\frac{2k}{k+1} = 2 - \epsilon$. Therefore, 2 is a tight lower bound on the speedup required to unsplittably schedule an admissible set of flows. In this section, we concentrate on *greedy* algorithms. We prove that any *greedy* algorithm can schedule an admissible set of flows when the speedup is at least 3.

Theorem 5: Any *greedy* algorithm can unsplittably schedule an admissible set of flows with a speedup $S \geq 3$.

Proof: Let F be the set of flow that cannot be scheduled using the *greedy* algorithm. Applying Lemma 4 to F, $B = 1$, and $S = 3$, we obtain that the blocking factor β satisfies $\beta > 2$. Consider a flow $f \in F$. Since f has a blocking link in every switch, and the blocking factor is more than 2, the amount of flow coming from the input port of f or going to the output port of f is more than $2k$. From the admissibility condition however, we know that at most $2k$ amount of flow can exist for any 2 ports. This is a contradiction. Therefore, the set F has to be empty. ■

The implication of Theorem 5 is that, with flows appearing and disappearing, a simple online algorithm can continue to schedule all flows, provided that at any time, the set of existing flows is admissible. Note that the online algorithm has to be *non oblivious* (see Section I for the definition of *oblivious*).

We can prove that 3 is actually a lower bound for two natural classes of *greedy* algorithms. We call these two classes *packing* and *load balancing*.

We begin by defining a *packing algorithm*:

Definition 3 (packing) A *packing* algorithm is a *greedy* algorithm by which, whenever possible, a new flow f is assigned to a switch s such that either $u(in_{s,f}) \neq 0$ or $u(out_{s,f}) \neq 0$.

For instance, a *greedy* algorithm that, whenever possible, does not utilize a link that is so far unutilized, is a *packing* algorithm. Similarly, the *greedy* algorithm that assigns a flow f to a switch s that maximizes $\max(u(in_{s,f}), u(out_{s,f}))$ is a *packing* algorithm.

Next, we define a *load balancing* algorithm:

Definition 4 (load balancing) A *load balancing* algorithm is a *greedy* algorithm by which, whenever possible, a new flow f is assigned to a switch s such that $u(in_{s,f}) = 0$ and $u(out_{s,f}) = 0$. If this is not possible, then f is scheduled in such a way to keep the maximum used link capacity, over all links, at a minimum.

For instance, the *greedy* algorithm that assigns a flow f to a switch s that minimizes $\max(u(in_{s,f}), u(out_{s,f}))$ is a *load balancing* algorithm.

We have the following results:

Theorem 6: There is no *packing* algorithm that can schedule any admissible set of flows with a speedup $S < 3$.

Proof: The proof is by choosing a speedup $S = 3 - \epsilon$ for any $\epsilon > 0$ and constructing an admissible set of flows that will cause the *packing* algorithm to fail in scheduling all the flows. We will assume that k is even and that $\frac{3k}{k+1} > S = 3 - \epsilon$, which can be obtained with a large enough k. We also require a large enough number N of input and output ports such that $N \geq k \times C^k_{k/2} + 2$. Let i_1, i_2, ..., i_N denote the input ports. Similarly let j_1, j_2, ..., j_N denote the output ports. For each $l = 1..N-1$, we receive k flows of size $\frac{k}{k+1}$ from input port i_l to output port j_l. Since $\frac{3k}{k+1} > 3-\epsilon$, at most 2 flows from i_l to j_l can be assigned to a single switch. Moreover, since the algorithm is a *packing* algorithm, once a flow from i_l to j_l is assigned to a switch s, the next scheduled flow from i_l to j_l will be assigned to switch s as well. Therefore, exactly $\frac{k}{2}$ switches will be used to schedule the k flows from i_l to j_l for $l = 1..N-1$. $C^k_{k/2}$ represents all possible ways of choosing $\frac{k}{2}$ switches among k switches. Since we have $N-1 = k \times C^k_{k/2} + 1$, at least $k+1$ (i_l, j_l) pairs will utilize the same $\frac{k}{2}$ switches s_1, s_2, ..., $s_{k/2}$. Let p_1, p_2, p_{k+1} be the input ports of these $k+1$ pairs. Now for $l = 1..k+1$, we receive a flow of size $\frac{k}{k+1}$ from p_l to j_N. Note that the admissibility condition still holds because $(k+1) \times \frac{k}{k+1} = k$. Since $\frac{3k}{k+1} > 3-\epsilon$, switches s_1, s_2, ..., $s_{k/2}$ cannot be used to schedule any of the new $k+1$ flows. Hence, exactly $\frac{k}{2}$ switches are avaialable to schedule the $k+1$ flows. Each of the available $\frac{k}{2}$ switches can hold at most 2 of the $k+1$ flows since $\frac{3k}{k+1} > 3-\epsilon$. Therefore, one of the $k+1$ flows cannot be scheduled. ■

Theorem 7: There is no *load balancing* algorithm that can schedule any admissible set of flows with a speedup $S < 3$.

Proof: The proof is similar to the one for *packing* algorithms. We assume that the number of ports is this time $N = 2N_0 + 1$, where $N_0 = (k \times C^k_{k/2} + 1)$. So for each $l = 1..N_0$, we can define the input ports i_l and i_{l+N_0}, and the output ports j_l and j_{l+N_0}. The idea is to make the *load balancing* algorithm utilize $\frac{k}{2}$ switches in the same way presented in the previous proof. We will illustrate how this can be done for one input-output pair (i_{l_0}, j_{l_0}). We first receive $\frac{k}{2}$ flows of sise $\frac{k}{k+1}$ from i_{l_0} to j_{l_0}. By the definition of the operation of a *load balancing* algorithm, these flows will be assigned to $\frac{k}{2}$ different switches, say s_1, s_2, ..., $s_{k/2}$. Now we receive $\frac{k}{2}$ flows of size $\epsilon < \frac{1}{k+1}$ from input port $i_{l_0+N_0}$ to ouput port j_{l_0}. The *load balancing* algorithm will schedule these flows using the other $\frac{k}{2}$ switches, say $s_{k/2+1}$, $s_{k/2+2}$, ..., s_k. Next we receive $\frac{k}{2}$ flows of size $\frac{1}{k+1}$ from input port $i_{l_0+N_0}$ to output port $j_{l_0+N_0}$. The *load balncing* algorithm will schedule the new flows using the switches s_1, s_2, ..., $s_{k/2}$. Next, we receive $\frac{k}{2}$ flows of size 1 from input port $i_{l_0+N_0}$ to output port $j_{l_0+N_0}$. If the *load balancing* algorithms assigns any of these new flows to any of the switches s_1, s_2, ..., $s_{k/2}$, then the maximum used link capacity will be $1 + \frac{1}{k+1}$. Therefore, the *load balancing* algorithm will schedule the new flows using switches $s_{k/2+1}$, $s_{k/2+2}$, ..., s_k, making the maximum used link capacity $1+\epsilon < 1+\frac{1}{k+1}$. Finally, we recieve $\frac{k}{2}$ flows of size $\frac{k}{k+1}$ from input port i_{l_0} to output port $j_{l_0+N_0}$. The *load balancing* algorithm will schedule these flows using the switches s_1, s_2, ..., $s_{k/2}$, making the maximum used link capacity $2\frac{k}{k+1}$ as opposed to $\frac{k}{k+1} + 1$ in case it assigns any of these flows to any of the switches $s_{k/2+1}$, $s_{k/2+2}$, ..., s_k. We can repeat this process for all $l = 1..N_0$ yielding to a situation similar to the one described in the previous proof. Note that the admissibility condition holds everywhere.

■

Future work will consider finding either a tight lower bound or a lower bound greater than 2 on the speedup required for any *greedy* algorithm to schedule any admissible set of flows.

References

[1] S. Mneimneh et. al., *On Scheduling Using parallel Input-Output Queued Crossbar Switches With No Speedup.* IEEE Workshop on High Performance Switching and Routing, HPSR 2001.

[2] S. Iyer et. al., *Making Parallel Packet Switches Practical.* IEEE/ACM INFOCOM 2001.

[3] Y. Dinitz et. al., *On The Single Source Unsplittable Flow Problem.* Combinatorica 19 (1) 1999.

[4] K-H. Tsai et. al., *Lower Bounds For Wide-sense Nonblocking Clos Network.* Theoretical Computer Science 261, 2001.

[5] D. Z. Du, et. al., *On Multirate Rearrangeable Clos Networks.* Siam Journal of Computing Vol. 28, No.2, 1998.

[6] Slepian, *unpublished manuscript.* 1958.