Scheduling Algorithms to Minimize Session Delays

Nandita Dukkipati and David Gutierrez

I. Introduction

A. Motivation

TCP flows constitute the majority of the traffic volume in the Internet today. Most of the traffic is carried by only a small number of flows (elephants), while the remaining large number of flows are very small in size or lifetime (mice). Provision of Quality-of-Service (QoS) to the flows is an important issue in the Internet. The long flows and short flows have different QoS requirements. In this paper, we are interested in the QoS provisioning for the short flows. Short connections expect relatively faster service than the longer connections i.e. an important measure of QoS for short flows is the response time as seen by a user. The response time as seen by a user is the time interval between the time of session initiation and the time when the last packet of the session is received. Henceforth, we will refer to this measure as the *session delay*. However, the Internet today just provides a *Best Effort* service to all flows and does not have any mechanism to provide QoS measures such as delay, throughput, packet loss and delay jitter. Thus, there is no mechanism in the Internet to reduce the response time for the short connections.

Thus, it is of interest to minimize the average response time for short connections in the network. The two fundamental pieces for QoS implementation in the network are:

1. QoS within a single network element (for example: buffer management schemes, scheduling algorithms and traffic shaping).

2. Techniques for coordinating QoS at each of the network elements to provide end-to-end QoS.

Schedulers at the network nodes are an important element of QoS provisioning. Scheduling algorithms are used to select packets for transmission on a link and hence control the bandwidth allocation among different flows at a node. The schedulers at different nodes in the network have to coordinate in order to provide an end-to-end QoS. There is no known optimal scheduling discipline to minimize the average session delays in either the single node case or the network case [1]. So, at this point the natural questions that arise are: How do the different scheduling disciplines proposed in the literature perform relative to each other, with respect to the average session delay in both the single node case and the network case? Is there an optimal scheduling discipline (i.e. a schedule that can minimize the session delays) under a known arrival traffic pattern? These questions motivate us to address the problem proposed in the following Section.

B. Problem

There are two main problems considered in this paper:

(1) Given a deterministic arrival traffic pattern, determine an *optimal schedule* in the single node case and the network case. An optimal schedule is one that minimizes the average session delay.

Generalized Processor Sharing (GPS) is a well studied scheduling algorithm in the literature and has very interesting bandwidth sharing properties. The next part of the problem in this paper is to use these properties of GPS to minimize average session delays.

(2) Given a network of GPS schedulers and a known arrival traffic pattern, find the weight (or ϕ) allocations for sessions at every node along their path, in order to minimize the average session delay.

C. Relation to Previous Work

It is known in the scheduling literature that *Shortest Remaining Processing Time* (SRPT) policy is an optimal discipline for the following scenario: Jobs arrive at a queueing system at arbitrary *known* times. They are served by a single server according to a preemptive policy. It is assumed that the server knows the service times of the jobs as soon as they enter the queue. For this scenario, it is proved in [3] that the SRPT policy minimizes the average delay of the jobs. It is however, not clear whether SRPT is an optimal schedule when "jobs" in the above setting are replaced by "sessions" (or flows). The main difference arises from the fact that in [3] an entire job arrives to the queueing system at a time instant, while in the queueing system considered in problem (1), a session arrives to the system with an arbitrary known arrival process. There is no work in the literature that addresses the problem of an optimal schedule under such a scenario either in the single node case or the network case.

Papers [1] and [2], which propose GPS, and other work in literature deal with the problem of obtaining a bound on per packet delay for sessions in GPS schedulers, *given* a particular weight allocation for the sessions. Problem (2) proposed above is different in two ways: First, the problem here, i.e. to find the weight allocations to minimize the session delays, is the inverse problem of the ones addressed in the literature. Second, previous work was interested in obtaining bounds on per packet delay. In our case we are interested in the total session delay. These two features differentiate the problem being addressed in this paper from all the previous work in the literature.

D. Research Contributions

The main contributions of this paper are:

• An optimal schedule for minimizing average session delays at a single node for 2 sessions.

• A heuristic schedule to obtain low average session delays in the case of N sessions at a single node.

• Weight allocation for sessions in Generalized Processor Sharing schedulers to realize a strict priority schedule, including the Shortest Job First schedule.

• A lower bound on average session delay in the multiple node case.

[1] Shortest Job First is an optimal discipline to minimize delays in the single node case, for a very specific arrival pattern. This will be elaborated further in the Section III

• A heuristic to schedule sessions to obtain a low average delay in the multiple node case.

II. Background

This Section provides a brief background on the scheduling policies used in this paper and introduces the definitions of some commonly used terms.
We assume a network of nodes where each node in the network uses a scheduling policy to schedule packets from the set of sessions arriving to the node. All results of this paper hold true for arbitrary, deterministic and *known* arrival processes for all the sessions. Throughout the paper, we consider fluid arrivals and fluid service. The extension to packet arrivals and packet service should not be difficult.
Definition 1: Session Delay: Let A_i be the arrival epoch of the first bit of the session i to the network and let D_i be the departure epoch of the last bit of the session i from the network. Then, the *Session Delay*, d_i, is defined as:

$$d_i = D_i - A_i$$

Definition 2: Average Session Delay: Let N be the number of sessions at a node (or the network in the multiple node case). Then, the *average session delay*, D, is defined as:

$$D = (\sum_{i=1}^{N} d_i)/N$$

Definition 3: Optimal Scheduling Policy: A scheduling policy at a node is said to be optimal under a particular arrival process if it minimizes the average delay of the sessions under that arrival process.
A Generalized Processor Sharing scheduler [1] serving N sessions is characterized by N positive real numbers, $\phi_1, ..., \phi_N$. These numbers denote the relative amount of service given to each session in the sense that if $W_i(\tau, t)$ is defined as the amount of session i traffic served by the GPS server during an interval $[\tau, t)$, then:

$$\frac{W_i(\tau,t)}{W_j(\tau,t)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, \ldots, N \tag{1}$$

for any session i that is continuously backlogged in the interval $[\tau, t]$. Thus, (1) is satisfied with equality for two sessions i and j that are both backlogged during the interval $[\tau, t]$. Note from (1) that whenever session i is backlogged it is guaranteed a minimum service rate of

$$r_i = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} C \tag{2}$$

where C is the capacity of the server.
Shortest Job First (SJF) is a non-preemptive policy that schedules the jobs in increasing order of their sizes. It provides the least average job delay among all non-preemptive scheduling disciplines. However, the SJF policy is optimal only when all the jobs are present at the server at once i.e. the jobs cannot arrive at arbitrary arrival epochs. There is no known non-preemptive policy which is optimal when the jobs can arrive at arbitrary arrival epochs.

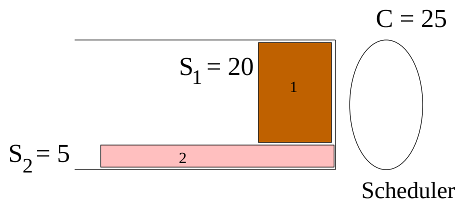

Fig. 1. Example to illustrate that SJF is not optimal when session traffic arrives at an arbitrary rate

The Shortest Remaining Processing Time (SRPT) is a preemptive policy that at all times processes that job, of those available, which has the shortest remaining processing time. The jobs could arrive to the system at arbitrary known times. It is assumed that the server knows the service times of the jobs as soon as they enter the queue. It is proved in [3] that with the SRPT discipline the number of jobs in the system at any point in time is less than or equal to the number of jobs in the system for any other rule simultaneously acting on the same sequence of arrivals and processing times. SRPT minimizes the average delay of the jobs.

III. Minimizing Delay in the Single Node Case

In this Section we address the problems (1) and (2) described in Section I-B for the case of a single node. We saw in Section II that the SJF policy is an optimal policy for minimizing the average delay of the jobs if all the jobs are present in the system at time $t = 0$. Now, consider a single node system in which all sessions start arriving at time $t = 0$. We introduce some notation here to describe the system:
• N: The total number of sessions in the system.
• $A_i(t)$: The arrival rate of the traffic of session i at time t. We assume that $A_i(t)$ is known for every session i and for all time t.
• S_i is the size of session i (in fluid units, ex. bits).
Then,

$$\int_0^\infty A_i(t)\, dt = S_i$$

A question that arises now is: Is SJF an optimal policy for the system described above?
The following example illustrates that SJF is *not* an optimal policy for the above system. The example has two sessions: Session 1 has a file size of $S_1 = 20$ and the entire file is present at the node at time $t = 0$. Session 2 has a file size of $S_2 = 5$ and starts arriving to the system at time $t = 0$ at a constant rate of 5 i.e. $A_2(t) = 5, 0 \leq t \leq 1$. The server capacity is $C = 25$. A SJF scheduler would give the highest priority to session 2 since it is the shortest job and would serve it at a rate equal to its arrival rate i.e. 5. The remaining capacity of the node, i.e. 20 is then allotted to session 2. Then, the total delay, D, under SJF = $d_1 + d_2 = 1 + 1 = 2$. On the other hand, serving session 1 completely before serving session 2 gives a delay of: $d_1 + d_2 = (20/25) + 1 = 1.8$. Clearly, there is no benefit in giving a higher priority to the shortest job in this example, since the shortest job is bottle necked by its arrival rate to the system. Thus, SJF is not an optimal policy in such a system.
The above example also serves to illustrate that the SRPT discipline, which is the preemptive version of SJF, is not optimal in

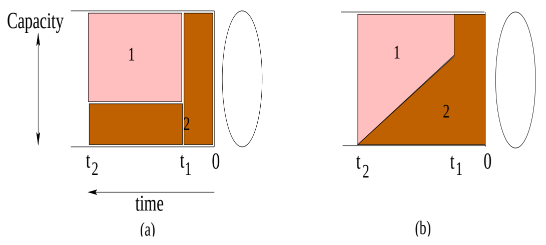

Fig. 2. (a) Optimal Schedule 1 (b) Optimal Schedule 2

the system of our interest. Another policy considered was Shortest Remaining Time First (SRTF), i.e. an SRTF scheduler would give the highest priority to the session that has the shortest time for its last bit to arrive completely to the scheduler. It can be seen through a simple example that such a policy is not optimal either.

A. Optimal Schedule for 2 Sessions

In this Section we identify an optimal schedule when there are two sessions at a single node. The arrival process of the sessions is arbitrary but *known*. The example in Figure III-A shows that an optimal schedule need not be a unique schedule. Consider the following arrival process: Session 1 has an arrival process as shown by region 1 in the Figure III-A(a) and session 2 has an arrival process as shown by region 2 in Figure III-A(b). Then, it is clear that both the schedules as shown in III-A(a) and III-A(b) are optimal. The delay of session 1 in both the schedules is $t_2 - t_1$ and that of session 2 is t_2.

The main motivation for the optimal schedule proposed here, is derived from the example in Figure 1. The optimal schedule must not only take the session size into consideration but also the session arrival process. This implies that if the shorter of the two sessions arrives over a very slow link, then it is possible that it is better for the scheduler to first devote its resources for the longer size session and then complete serving the shorter session by the time the last bit of the shorter session arrives. This motivates us to define a scheduling policy called *Shortest Time to Finish First* (STFF).

Definition 4: Session i is said to have a higher priority than session j in a schedule, if the session j is never served as long as session i is backlogged. In other words, session i is always given as much capacity as it can use, and the remaining capacity is used to serve session j.

The basic idea of the STFF algorithm is to prioritize the sessions according to the minimum time that they would take to finish, i.e. the time that each session would take to complete if it were allotted the entire server capacity, C. The STFF algorithm for the case of 2 sessions is described below:

```
STFF:
Step 1: Compute the minimum time to finish for every session
$R_i(t)$ = maximum service rate session $i$ can receive at time $t$
$S_i$ = Size of session $i$
$TF_i = min\{t : \int_0^t R_i(y)dy = S_i\},\ i = 1, 2$
Step 2: Determine the schedule
if $TF_1 < TF_2$
then session 1 has a higher priority then session 2;
else session 2 has a higher priority than session 1.
```

As an example; consider the example in Figure 1. The STFF schedule gives a higher priority to session 1, because $TF_1 = 0.8$ is less than $TF_2 = 1$. Thus, it serves session 1 before serving session 2, which is optimal in this case. The following Lemma proves the optimality of STFF in the two session case.

Lemma 1: The STFF discipline minimizes the average delay of two sessions at a single node server.

Proof: The following notation is used:
•$\delta_i(t)$ = fraction of the server capacity devoted to session i at time t,
$\delta_i(t) = 0$; for $t <$ arrival time of session i,
$\delta_i(t) = 0$; if there are no flow i bits in the server at time t,
•S_i: Size of flow i
•C_i : Completion time of flow i, i.e. the time when the last bit of flow i departs from the system

$$C_i = min\{t : \int_0^t \delta_i(y).Cdy = S_i\}$$

The superscript, o, is used to identify measures related to any arbitrary schedule and the superscript, s, is used to identify the measures related to the STFF schedule. No superscript is used for a statement that is true for both schedules.
Clearly, $C_i + C_j = min(C_i, C_j) + max(C_i, C_j)$. Also: $max(C_i^o, C_j^o) =$

$$\begin{aligned} &= min\{t : \int_0^t (\delta_i^o(x) + \delta_j^o(x)).Cdx = S_i + S_j\} \\ &= min\{t : \int_0^t (\delta_i^s(x) + \delta_j^s(x)).Cdx = S_i + S_j\} \\ &= max(C_i^s, C_j^s) \end{aligned} \tag{3}$$

i.e. regardless of how i and j are scheduled from $(0, \infty)$, the last session will always finish at the same time, given that the system is work-conserving. On the other hand, $min(C_i, C_j)$ is minimized if the sessions are scheduled according to STFF. This follows from the definition of STFF, i.e. the function:

$$min\{t : \int_0^t \delta_k(x).Cdx = S_k\}$$

is minimized by choosing k as the session which has the shortest time to finish. Therefore,

$$min(C_i^s, C_j^s) \leq min(C_i^o, C_j^o)$$

It follows that: $max(C_i^s, C_j^s) = max(C_i^o, C_j^o)$ and $min(C_i^s, C_j^s) \leq min(C_i^o, C_j^o)$. Therefore the STFF discipline in the case of 2 sessions is optimal.

B. Schedule for N Sessions

There could be multiple schedules which are optimal in the N session case. Let $\{S^{opt}\}$ be the set of all optimal schedules. Let s be an optimal discipline that $\in \{S^{opt}\}$, and i and j be any two sessions. Let $\delta_i^s(t)$ be the fraction of server capacity occupied by session i at time t. Then, the capacity occupied by the two sessions at time t is: $(\delta_i^s(t) + \delta_j^s(t)).C$. Let $\sigma_{ij}^s(t) = (\delta_i^s(t) + \delta_j^s(t)).C$. Let $\sigma_{ij}^s = \{\sigma_{ij}^s(t)\}$, i.e. σ_{ij}^s is a function of time that represents the bandwidth occupied by the sessions i and j, in the schedule s.

Definition 5: A scheduling discipline, s, is said to be pairwise STFF, if for every pair of sessions, i and j, the departure epochs of sessions i, j are the same as those if STFF were practiced for sessions i, j within the bandwidth function σ^s_{ij}, of the schedule s.

The following Lemma identifies a characteristic of the optimal solution in the case of N sessions at a node.

Lemma 2: Every optimal discipline in $\{S^{opt}\}$ must be pairwise STFF.

The proof of this Lemma follows easily from Lemma 1. If an optimal schedule s is not pairwise STFF, then there exist a pair of sessions i and j, which when rescheduled within the bandwidth σ^s_{ij}, according to the STFF discipline, will produce a more optimal solution (by Lemma 1). This contradicts the assumption that s is an optimal discipline.

The following algorithm extends the STFF algorithm for the case of N sessions.

STFF-N:
Step 1*: Initialization*
N : Set of all sessions
N_p : Set of all sessions whose priorities have been decided; $N_p = \emptyset$
Step 2*: Loop to prioritize the sessions*
for $j \leftarrow 1$ to $|N|$
Step 3*: Compute the minimum time to finish*
$TF_i =$ Time to completion of session i if the remaining unallotted capacity were fully available for use; $\forall i \in \{N - N_p\}$
Step 4*: Assign the priority j* to session k, where
$TF_k = min_{i \in (N-N_p)} TF_i$;
Step 5*: Update N_p and remaining capacity*
$N_p = N_p \cup \{k\}$;
$C(t) = C-$ (capacity taken by sessions $\in N_p$) $\forall t$; C is the total capacity

The basic idea of the algorithm is to compute the minimum time that each of the sessions would take to complete if they were given as much capacity as available, then assign the session which has the minimum time to completion the highest priority. A prioritized session is no longer considered in the subsequent iterations of the algorithm. The capacity that is taken by the prioritized sessions is subtracted, to get the available capacity. The algorithm goes on to the subsequent iteration, and computes the minimum time to finish for the remaining sessions with the new available capacity. On termination of the algorithm, each session is assigned a priority. The final schedule then serves the sessions with these priorities.

The next question that arises is: Is the $STFF - N$ discipline optimal? Figure 3 illustrates an example to show that STFF-N is not an optimal discipline. In this example, there are three sessions. Session 1 is present completely in the system at time $t = 0$ and takes time $t_1 - \delta$ to complete when assigned the full server capacity. Sessions 2 and 3 arrive to the server at a rate each of $C/2$ from time $t = 0$ to $t = t_1$. The STFF-N schedule (shown in Figure 3(a)) gives the highest priority to session 1, since it has the minimum time to completion. It serves sessions 2 and 3 after completing serving session 1. Therefore, the delay of each session is: $d_1 = t_1 - \delta$; $d_2 = (t_1 - \delta) + t_1/2$ and $d_3 =$

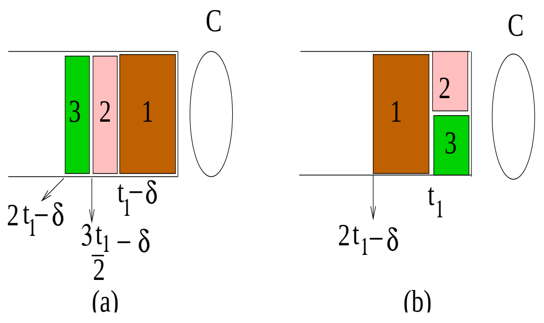

Fig. 3. (a) STFF-N schedule (b) Optimal schedule

$(3*t_1/2 - \delta) + t_1/2$. That gives a total delay of : $D_{STFF-N} = 9/2 * t_1 - 3 * \delta$, while if the sessions 2 and 3 are scheduled before session 1, as shown in Figure 3(b), then the total delay is: $D_o = t_1 + t_1 + (2 * t_1 - \delta) = 4 * t_1 - \delta$. If δ is chosen small enough then, $D_o < D_{STFF-N}$.

The solution in Figure 3(b) is the optimal solution for the above example. Note, that the pairwise STFF property is satisfied in both schedules. Thus, *pairwise STFF property is a necessary but not a sufficient condition for optimality*. The priority assigned to the sessions based on the shortest time to finish, when $N > 2$, is not necessarily the optimal priority. Does there exist a strict priority of the sessions that yields an optimal schedule? This remains an open question.

IV. Minimizing Delays in a Generalized Processor Sharing Scheduler

In Sections III-A and III-B, we were interested in defining an optimal priority based schedule for minimizing the average session delay. In this Section, we assume a Generalized Processor Sharing (GPS) scheduler in a single node system. The problem addressed is: *Given a set of N sessions, each with an arbitrary but a known arrival process, what is the weight allocations for these sessions in a GPS scheduler that would minimize the average session delay*?

As a starting point, first consider the simple scenario, when the entire traffic of all sessions is present in the server at time $t = 0$. In this case we know that serving the sessions in the increasing order of the file sizes, or practicing the shortest job first schedule is optimal. So, the question now is: Can we assign weights in GPS, such that the sessions are served in the increasing order of the file sizes ? It is possible to do so, and the method is shown below.

Let $i_1, i_2, \ldots, i_N$ denote the sessions in increasing order of session sizes. Then, the problem is to find weights $\phi_1, \phi_2, \ldots, \phi_N$ such that:

$$\frac{\phi_{i_1}}{\sum_{k=1}^{N} \phi_{i_k}}.C = C - \delta \quad (4)$$

$$\frac{\phi_{i_2}}{\sum_{k=2}^{N} \phi_{i_k}}.C = C - \delta \quad (5)$$

$$\vdots$$

$$\frac{\phi_{i_{N-1}}}{\sum_{k=N-1}^{N} \phi_{i_k}}.C = C - \delta \quad (6)$$

$$\sum_{k=1}^{N} \phi_{i_k} = 1 \tag{7}$$

$$\phi_i > 0; i = 1, \ldots, N \tag{8}$$

In the above equations δ can be chosen arbitrarily small, but positive. δ has to be > 0 because the weights of all the sessions are positive and therefore if a session i_j is being served then all the sessions with file size greater than that of i_j also receive a small amount of service, which is δ. The above set of simultaneous equations can be solved to obtain the ϕ_i values. As an example:
$N = 2 : \phi_{i_1} = 1 - \frac{\delta}{C}; \phi_{i_2} = \frac{\delta}{C}$
$N = 3 : \phi_{i_1} = 1 - \frac{\delta}{C}; \phi_{i_2} = (1 - \frac{\delta}{C})\frac{\delta}{C}; \phi_{i_3} = (\frac{\delta}{C})^2$
In general, for N sessions, the solution is:

$$\phi_{i_j} = (1 - \frac{\delta}{C})(\frac{\delta}{C})^{j-1}; 1 \leq j \leq N-1$$

$$\phi_{i_N} = (\frac{\delta}{C})^{N-1} \tag{9}$$

The next question of interest is to find the difference in the total delay obtained by SJF and the delay obtained by GPS with the above ϕ allocations. As before, let $i_1, i_2, \ldots, i_N$ be the ordered set of sessions (in increasing order of the file sizes). Let S_{i_j} be the file size and d_{i_j} be the delay experienced by session i_j in SJF scheduler. Then: $d_{i_1} = \frac{S_{i_1}}{C}, d_{i_2} = d_{i_1} + \frac{S_{i_2}}{C}, \ldots, d_{i_N} = d_{i_{N-1}} + \frac{S_{i_N}}{C}$. The total delay of the sessions under the SJF scheduler is:

$$D_{SJF} = N * S_{i_1} + (N-1) * S_{i_2} + (N-3) * S_{i_3} + \ldots + S_{i_N}$$

The total delay experienced by the sessions in GPS with the weight allocation obtained in (9), can be computed with a little algebra. For example:
$N = 2; d_{i_1} = \frac{S_{i_1}}{C-\delta}; d_{i_1} = \frac{S_{i_1}+S_{i_2}}{C}$
$N = 3; d_{i_1} = \frac{S_{i_1}}{C-\delta}; d_{i_2} = \frac{S_{i_1}}{C} + \frac{S_{i_2}}{C-\delta}; d_{i_3} = \frac{S_{i_1}+S_{i_2}+S_{i_3}}{C}$
The general expression for the total delay in GPS, after rearranging some terms, for N sessions is:

$$\begin{aligned} D_{GPS} &= (N-1) * \frac{S_{i_1}}{C} + (N-2) * \frac{S_{i_2}}{C} + \ldots + \frac{S_{i_{N-1}}}{C} \\ &+ \frac{S_{i_N}}{C} + \frac{S_{i_1}}{C-\delta} + \frac{S_{i_2}}{C-\delta} + \ldots + \frac{S_{i_{N-1}}}{C-\delta} \end{aligned} \tag{10}$$

The difference between the delays is:

$$D_{GPS} - D_{SJF} = \frac{\delta}{C * (C-\delta)} \sum_{i=1}^{N-1} S_i$$

Thus, if the number and the total size of the sessions is fixed, then δ can be chosen arbitrarily small in order to have a negligible difference between the delays of GPS and SJF schedulers.
SJF is an example of a strict priority scheduler. In SJF the sessions have a strict priority based on the size of the sessions. If $i_1, i_2, \ldots, i_N$ is some other priority of the given sessions, then the weight allocation obtained in (9) can be used in GPS to realize these priorities. *Therefore, GPS can realize any strict priority based schedule.*

In the general scenario, when the arrival process of the sessions is arbitrary, we saw in Section III-B that choosing an optimal priority for the sessions, that minimizes the total delay, is not trivial. However, if such a priority exists, then the GPS weights can be chosen as in (9), to realize the priority. If a higher priority session cannot take the entire bandwidth, then GPS distributes the remaining bandwidth to the lower priority sessions in proportion to their weights. As a result, the next higher priority session gets the major portion of the remaining bandwidth. This is possible due to the nice bandwidth sharing properties of GPS and the choice of the above weights.

V. Minimizing Delay in the Network Case

Minimizing average session delays in the network case is similar to the Job Shop scheduling problem, which is NP-hard. In this Section we propose heuristics to schedule the sessions in the network case in order to obtain a low average session delay. Simply extending scheduling techniques that provide good results for the single node case described above is not enough for the multiple node case, since these techniques do not use any information on how the session flows interact with each other at different nodes throughout the network.

A. A Lower Bound

In this Section we obtain a lower bound for the average delay of the sessions in the network scenario. The lower bound is obtained as follows:
1. N: Set of sessions. For every session $i \in N$ identify the bottleneck node (the node with the least capacity) along its path.
2. K : Set of nodes. For every node $k \in K$:
R_k = Set of sessions bottle necked at node $k; k \in K$
$F_k = \sum_{i \in N} \sum_{j:S_j <= S_i} S_j; \forall k \in K$ (Total session delay in SJF scheduler)
3. Minimum Average Session Delay $\geq \sum_{k \in K} F_k / N$
This bound holds true when all the sessions start at the same time. The basic idea is that, if a set of sessions are bottle necked by a node then the best possible way that they can be served by that node would be if all the sessions were to be present entirely at the node, and the node practices SJF schedule on them.

B. Heuristic Schedule

In this Section we present a heuristic schedule that provides a strict priority at every node for every session that traverses it. The following example illustrates that *there need not exist any strict priority of the sessions at every node that would yield an optimal schedule.*

In Figure 4, denote the three sessions by x_1, x_2 and x_3. In this example, x_1 goes through node 1, x_2 goes through nodes 2 and 1, and x_3 goes through nodes 2 and 3. Table I shows four ways in which the priorities can be assigned at nodes 1 and 2, and the corresponding total delay:

However, if the network nodes were to use the following scheduling, the average session delay would be less:
1. Node 1 serves session 1 until $t = 10/2 = 5$.
2. Node 2 serves session 3 during the same period, thus keeping node 3 busy.
3. Node 2 servers session 2 after $t = 5$.

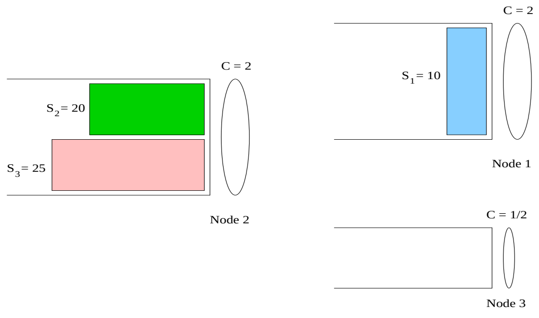

Fig. 4. There does not exist an assignment of priorities that yields an optimal schedule

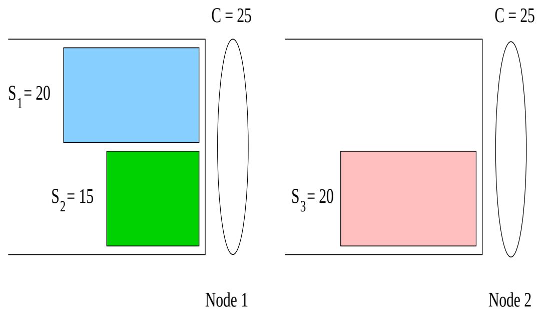

Fig. 5. Two node network with three session flows

Table I

Priorities at Node 1	Priorities at Node 2	Total Delay	Average Delay
$x_1 > x_2$	$x_2 > x_3$	80	26.67
$x_1 > x_2$	$x_3 > x_2$	77.5	25.83
$x_2 > x_1$	$x_2 > x_3$	85	28.33
$x_2 > x_1$	$x_3 > x_2$	77.5	25.83

With this schedule, the delay for x_1 is $10/2 = 5$, for x_2 is $5 + 20/2 = 15$ and for x_3 is $25/(1/2)$. Thus, the total session delay is 5 + 15 + 50 = 70 and the average session delay is 23.33, which is less than any of the possible priority based solutions. However, defining a schedule based on strict priorities is simpler than any other schedule and that is why we are considering them.

Priorities throughout the network nodes may be assigned in many different ways: a particular flow could have a higher priority with respect to another flow at a particular node and a relatively lower priority at another node. We define a network wide priority, as one in which the relative priority for every two flows is the same throughout all the nodes of the network. Equivalently, a unique priority could be assigned for every session flow on the whole network and every node could then simply apply the priorities for the sessions that traverse it.

The Shortest Job First priority is an example of a network wide priority. In the SJF priority, the shortest job in the network has the highest priority at every node, the second shortest job has the next highest priority and so on. However, as the example below shows, using SJF is not the best way to assign priorities, since the interaction among sessions at other nodes is being ignored in this way of assigning priorities. That is, the fact that serving a particular flow is reducing the available capacity at all nodes along its path is being ignored.

For example, suppose we have a network of two nodes and three flows as in Figure 5. Session 1 needs to go only through node 1, session 2 needs to go through both nodes 1 and 2 and session 3 needs to go only through node 3. If a Shortest Job First priority is used in this case, session 2 will have the highest priority at both nodes. Thus, both session 1 and session 3 will be prevented from service at both nodes until session 2 is finished. Thus, the completion time of the sessions will be $C_2 = 15/25 = 0.6, C_1 = C_2 + 20/25 = 0.6 + 0.8 = 1.4, C_3 = C_2 + 20/25 = 1.4$. The average session delay will be $(C_1 + C_2 + C_3)/3 = 3.4/3 = 1.13$. If the priorities had been assigned considering the blocking information, i.e. the fact that session 2 blocks session 1 at node 1 and session 3 at node 2, the average session delay would be lower, as we will see below.

The heuristics we propose includes the *blocking information* of every session flow by calculating its priority according to the time it would take to traverse the network if the available bandwidth was dedicated exclusively to servicing it, plus the additional delay that other sessions may incur if that particular session was given a higher priority. The algorithm works based on the premises that flows that have the shortest time to finish and that block other sessions the least should be given a higher priority.

We will use the following notation:

Denote the set of all sessions to be serviced by $N = x_1, x_2, ..., x_N$

Definition 5: Time to Finish: The time to finish for a particular session x_i is denoted by $TTF(x_i)$ and it refers to the amount of time that it would take the network to service the session if all of it's available capacity was dedicated exclusively to that session.

Definition 6: Blocking Delay: In a set of sessions whose priority has not yet been assigned, the blocking delay of a particular session x_i on another session x_j refers to the amount of time by which the delay of session x_j would increase if x_i had a higher priority than x_j. We will refer to this extra total network session delay of x_j caused by x_i (if x_i had a higher priority than x_j) as $d(x_i, x_j) \geq 0$. The algorithm described below will compute the blocking delay among all the sessions that have not yet been prioritized, while deciding on their appropriate priority.

Algorithm:

Denote the set of session flows whose priority has been decided as N_p. Note that the set of session flows whose priority has not yet been assigned is $\{N - N_p\}$.

On each iteration, the algorithm should consider all the sessions whose priority is still unassigned. For each of the unassigned sessions, the algorithm calculates their corresponding TTF given the available network capacity (that is the total network capacity minus the capacity used by the sessions whose priority has already been assigned), and their corresponding blocking delay with all the other unassigned sessions. The sum of TTF and blocking delay for each session gives the total cost of a session. The algorithm then picks the session with minimum total cost and includes it in N_p with the next available priority. In other words, on each iteration the algorithm will assign a priority for one session. This procedure is repeated until all the sessions have been assigned a priority. It's straightforward to see that this algorithm is $O(N^3)$.

Pseudo-code:

```
Step 1: Initialization
$N$: Set of all sessions
$N_p = \emptyset$: Set of all sessions whose priorities have been decided
Step 2: Loop to prioritize the sessions
for $j = 1$ to $|N|$
Step 3: Calculate the total blocking delay, time to finish and total cost for every session, $\forall x_i \in \{N - N_p\}$
$blocking\_delay(x_i) = \sum_j d(x_i, x_j)$
$TTF(x_i)$ = time to process $x_i$ with the current network capacity (that is, the total network capacity minus the capacity taken by sessions $\in N_p$.
$F(x_i) = TTF(x_i) + blocking\_delay(x_i)$
Step 4: Take the session with the least cost and assign it the priority $j$
find the $m$ for which $F(x_m) <= F(x_i) \forall x_i \in \{N - N_p\}$
priority($x_m$) $= j$;
include $x_m$ in $N_p$
```

Consider again the example in Figure 5. The network wide priorities using the above algorithm are, from highest priority to lowest, x_1, x_3, x_2. The average delay when these priorities are used: $C_1 = 20/25 = 0.8, C_3 = 20/25 = 0.8$ and $C_2 = max(C_1, C_2) + 15/25 = 0.8 + 0.6 = 1.4$. So, the average session delay will be $(C_1 + C_2 + C_3)/3 = 3/3 = 1$.

Note that if the SJF or STTF criteria had instead been used at every node, as we saw before, x_2 would have had a higher priority at both nodes than x_1 and x_3, even though it's blocking delay cost is high. As a result, the average session delay is $3.4/3 = 1.13$, which is greater than the one we get using the algorithm. Even though the improvement on average session delay may not seem too much in this example, it can be inferred that in more complex networks, as more session flows share the same nodes, the blocking factor becomes more important and greater gains may be obtained.

If we used this priorities and assigned GPS weights as described in section IV-A, we can see how a network of GPS schedulers will provide the same or better average session delay than a network of SJF or STTF schedulers.

VI. Summary and Future Work

In this paper, we observed that in the single node case SJF is not an optimal schedule when sessions have arbitrary arrival traffic pattern. We then devised a scheduling algorithm, Shortest Time to Finish First (STFF), and proved that it minimizes the average session delay, in the case of two sessions at a single node. However, we showed that STFF is not an optimal schedule in the general N session case.

We also presented a method for weight allocation in GPS schedulers to realize strict priority schedulers. This implies that GPS schedulers can realize the Shortest Job First schedule, both in the single node and the multiple node case. So, a network of GPS schedulers is more generic than a network of SJF schedulers.

In the multiple node case, we showed that there need not exist a strict priority schedule of sessions that would also be an optimal schedule. We also obtained a lower bound on the average session delay in the multiple node case. Finally, we presented a heuristic schedule to obtain low average session delays in the network scenario.

Directions for future research include:

• Does there exist a strict priority schedule of sessions that would also be an optimal schedule in the case of N sessions at a single node ?

• Is the problem of finding an optimal schedule in the single node case when there are N sessions with arbitrary arrival traffic pattern, an NP-hard problem?

• Obtain tighter lower bounds on the average session delay in the multiple node case.

• Compare the session delays obtained by the heuristics with the lower bounds and improve the heuristics, in both the single node and the multiple node case.

VII. Acknowledgments

We thank Rui Zhang for some helpful discussions, specially in the single node case.

References

[1] A.K Parekh and R.G. Gallager, A Generalized Processor Sharing Approach to Flow Control in Integrated Services Packet Networks: The Single Node Case, IEEE/ACM Transactions on Networking, vol.1, No. 3, pp. 344-357, June 1993
[2] A.K Parekh and R.G. Gallager, A Generalized Processor Sharing Approach to Flow Control in Integrated Services Packet Networks: The Multiple Node Case, IEEE/ACM Transactions on Networking, vol.2, No. 2, pp. 137-150, April 1994
[3] Linus Schrage, A proof of the optimality of the shortest remaining processing time discipline Operations Research, 16, 687-690