Scheduling High-Rate Sessions in Fractional Lambda Switching Networks: Algorithm and Analysis

Thu-Huong Truong[*] , Mario Baldi[**], Yoram Ofek[*] *IEEE Fellow*
[*] *University of Trento, Italy - {huong.truong,ofek}@dit.unitn.it*
[**] *Politecnico di Torino, Italy - mario.baldi@polito.it*

Abstract

This work addresses the high-rate session scheduling problem in Fractional Lambda Switching (FλS) networks. With its global phase synchronization and pipeline forwarding (PF) operation, FλS offers promising network performance and scalability over its competitors, e.g., Time Division Multiplexing (such as SONET/SDH) and Wavelength Division Multiplexing (WDM). Yet, Non-Immediate Forwarding (NIF) brings challenging complexity to session scheduling, where other known scheduling methods (e.g. RWTA) are not applicable. A forwarding graph is used to wholly examine the huge schedule space for an end-to-end high-rate NIF session. An efficient scheduling algorithm, eSSM, is proposed to explore all possibilities on the graph and present the optimized non-blocking schedule. Complexity bounds are then devised analytically and experimentally verified under specific circumstances. A low-complexity heuristic is proposed to avoid the complexity of eSSM in low-load networks.

1 Introduction

Fractional Lambda Switching (FλS) [1]-[3] can provide the reliability, fast transport, minimum jitter, and flexible bandwidth provisioning required in modern optical networks. FλS switches and forwards data in temporal units, called Time Frames (TF), on WDM channels similarly to the well-known TDM-WDM (Time Division Multiplexing-Wavelength Division Multiplexing) solutions, such as Time-shared Wavelength-Routed (WR) networks [4]. However, FλS deploys global phase synchronization — achieved for example by using UTC time as provided by GPS to devise the TF duration — and the pipeline forwarding (PF) of packets in pre-allocated TFs [1]. These features improve network performance, reduce the system complexity and increase scalability. Yet, issues related the FλS control plane, specifically TF scheduling, need to be studied.

According to the PF principle, packets are forwarded in TFs (as "virtual containers") in a "lock-step" manner across the route at a predefined scheduled request. PF can be performed in two modes: (1) immediate forwarding (IF) or (2) non-immediate forwarding (NIF). NIF is more interesting as it enables greater flexibility and higher utilization. Yet, scheduling with NIF is more challenging due to its huge solution space leading to a potentially high complexity in finding a solution [7]. There are some existing scheduling results in TDM-WDM networks for various scenarios (e.g., multi-rate, online/offline) based on network throughput optimization [5][6]. However, those scheduling methods only address scenarios equivalent to the IF case of FλS where once the scheduling problem has been solved at one switch, the schedule at any switch on the route is directly determined.

Scheduling in NIF-FλS for a fixed basic-rate (1-TF) session was first proposed in [7], where . This paper addresses the case of a high-rate request of any number of TFs on a homogeneous single-wavelength optical network. Section 2 summarizes the FλS principles, the scheduling model and formulates the problem. Section 3 shows the way to construct a forwarding graph for the multiple-TF-request scheduling model and proposes an algorithm, named **eSSM** (**e**fficient **S**urvivor-based **S**earch for a **M**ultiple-TF session) to solve the scheduling problem. Section 0 presents a heuristic to be used with low network load as in such scenario eSSM complexity is high. Section 5 discusses a variant of the scheduling algorithm that avoids out-of-order packet delivery. Conclusions are drawn in Section 6.

2 Scheduling problem formulation

2.1 Network principle: timing and pipeline forwarding

Timing: In FλS networks, timing structure is constructed in a way that multiple time frames (TFs) are grouped into a time cycle (TC) and multiple TCs are equal to a standard UTC. The timing and synchronization is globally provided with UTC systems (GPS, Gallileo…) [1].

Pipeline-forwarding (PF): the fundamental principle of FλS networks is that packets are pipeline-forwarded

in TFs with a predefined forwarding schedule that is responsive to UTC timing and without header processing. The necessary condition for PF is having propagation delay as an integer number of TFs [1]. In order to realize this all incoming TFs should be aligned with UTC. Without loss of generality, in this work we assume the availability of this UTC alignment operation and ignore the propagation delay as well as the alignment delay.

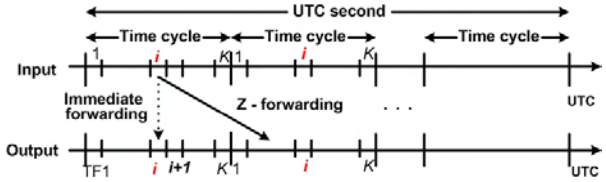

Figure 1 – Timing structure and pipeline forwarding

Z-forwarding delay is the delay due to intentionally holding (buffering) the incoming TF (i.e.: its packets) for a duration of $0 \to Z$ TFs before forwarding to the next switch on the route. The *Z*-forwarding has two cases, as shown in Figure 1:

1. *Z=0* – Immediate Forwarding (IF): incoming TFs are forwarded with zero delay to next switch.
2. *K>Z>0* – Non-Immediate Forwarding (NIF): incoming TFs can be forwarded to next switch with delay being any from 0 to *Z* TFs.

NIF obviously increases the possibility to accommodate more concurrent traffic flows, compared to using IF, thus reducing blocking probability and increasing network utilization

TF provision: The scheduling of TFs for traffic flows is periodical with a cycle, normally a TC, after which a schedule is repeated . The TFs inside a TC therefore are the bandwidth units to be provisioned to the sessions.

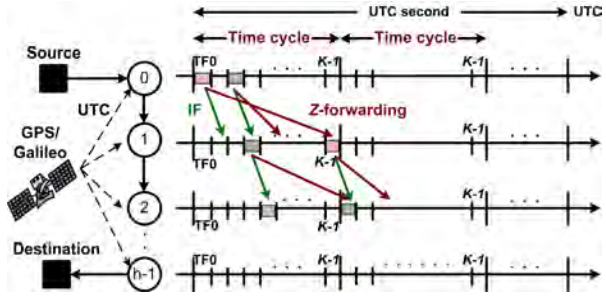

Figure 2 – Network model for a 2-TF session

2.2 Scheduling problem

2.2.1 Network model:

A scheduling problem is considered for a single high-rate session on a network route, without dependency on the topologies of the real networks. The route carries traffic from Source to Destination via *h* switches. The session requires multiple TFs — hereafter called a *g*-TF session — in a single-channel, homogeneous FλS network. Homogeneity means that the same bit rate and TF structure are deployed on all links.

The scheduling problem is studied (1) on one predefined route (i.e. route selection completed), (2) without considering propagation delay and UTC alignment delay, and (3) allowing TFs of the scheduled session to be non-contiguous.

2.2.2 g-TF Scheduling Problem Formulation:

Beside the extensively studied blocking problem [1][2], scheduling in a scenario with requested data rate of multiple TFs is a complex task due to a huge space of possible schedules, as analyzed in the following.

Definitions

Available TF - a TF at an output of a switch during which packets of the requesting flow can be transmitted.

Choice - a set of *g* available output TFs selected for a given *g*-TF bandwidth request at switch *j*

$$C_k^j = \left\{\left(TF_i^{\,j}\right)_l\right\} \quad i.e: \quad C_k^j(l) = TF_i^{\,j}$$

Where:

$TF_i^{\,j}$: Available TF *i* at switch *j* (*i*: 0→*K*-1, *j*:0→*h*-1)

$C_k^j(l)$: *l*-th position (*l*: 0→*g*-1) in a TF set of k^{th} choice

A choice is constrained by the rule of *valid compound forwarding*:

Valid compound forwarding - a compound forwarding of data from a choice $C_{k'}^{j-1}$ to a choice C_k^j is valid **iff** all its single forwards are valid. A single forward is valid according to *Z*-forwarding when

If $TF_{i'}^{\,j-1}$ is selected for $C_{k'}^{j-1}(l)$,

Then $C_k^j(l)$ can only be a free $TF_i^{\,j}$ in the range of $[i', (i'+Z) \bmod K]$ (in the same or the next TC).

i.e. $[i-i']_l \bmod K \le Z$ for every *l-th* position

Schedule - a schedule is a sequence of choices over a predefined route of multiple switches

$$S = \left\{\ldots, C_{k'}^{j-1}, C_{k'}^{j}, \ldots\right\}$$

Observation

For Z-forwarding (0<Z<K-1), the total number of possible schedules (S_Σ) for a g-TF flow over an h-hop route is bounded as:

$$\frac{K!}{(K-g)!}\cdot\left(\frac{(Z+1)!}{(Z+1-g)!}\right)^{h-1} < S_\Sigma < \frac{K!}{(K-g)!}\cdot(z+1)^{g(h-1)} \quad \text{(i)}$$

Proof:

At switch 0: the number of possible choices is equal to the number of different sets of g TFs, which are permutations without repetition of g TFs. Therefore, there are $G_0 = \frac{K!}{(K-g)!}$ choices to forward data to the next switch.

At switch 1: each of the G_0 choices evolves to G_1 choices which cannot be greater than $(Z+1)^g$ nor smaller than $\frac{(Z+1)!}{(Z+1-g)!}$. We derive that by considering two border cases.

The upper bound is derived by considering the set of TFs that can be used to forward from each single TF of a choice as non overlapping with the TFs that can be used for forwarding from the other TFs of the choice. Since packets arriving during each TF can be forwarded during (Z+1) possible TFs and there are g TFs in a choice, the total number of possible choices is

$$G_1^{up} = \underbrace{(Z+1)\cdot(Z+1)..(Z+1)}_{g} = (Z+1)^g \cdot$$

The lower bound is derived assuming maximum overlap between the TFs that can be used to forward packets arrived at switch 1 during various TFs of a choice. Maximum overlap is achieved when the TFs of a choice are contiguous. There are (Z+1) options for choosing a forwarding TF for packets arriving during the first TF of a choice, Z for the second TF, and so on. Considering the g^{th} TF in the choice, (Z+1-g+1) TFs can be chosen for forwarding packets. Consequently, the total number of possible choices at switch 1 in this case is:

$$G_1^{low} = (Z+1)\cdot Z \cdot ... \cdot (Z+1-g+1) = \frac{(Z+1)!}{(Z+1-g)!}$$

Choices evolve the same way as in switch 1 at every subsequent switch on the path. Consequently, the total number of possible schedules S_Σ at the last switch (h-1) is given by combination of all the single hop possible choices: $S_{low} < S_\Sigma < S_{up}$

Where:

$$S_{low} = G_0 \cdot G_1^{low} \cdot ... \cdot G_{h-1}^{low} = \frac{K!}{(K-g)!} \cdot \left(\frac{(Z+1)!}{(Z+1-g)!} \right)^{h-1}$$

$$S_{up} = G_0 \cdot G_1^{up} \cdot ... G_{h-1}^{up} = \frac{K!}{(K-g)!} \cdot (Z+1)^{g(h-1)}$$

□

The space of possible schedules S_Σ bounded by (i) grows exponentially with the length of the path h, which is a limitation for the network scale. Moreover, also the amount of bandwidth required by a session (i.e., the number of TFs g to be scheduled) exponentially affects the size of the solution space. Therefore, an efficient algorithm is to be devised to perform scheduling so that the time required to set up a reservation for a new session an be limited.

3 Scheduling algorithm: design and complexity analysis

The algorithm proposed in this paper and the analysis of its complexity are based on a graph that represents all scheduling possibilities. Consequently, the remainder of this section is first introducing such graph, then presenting the algorithm, and finally analyzing its complexity.

3.1 Forwarding graph construction

A trellis graph is used to represent the scheduling possibilities by mapping each choice on each link onto a vertex. The resulting graph is called *forwarding graph*. When a single TF schedule is to be found, each vertex represents one TF at a link [7]. Two arguments are provided to motivate the way the graph is constructed.

Argument 1

Vertices are permutations without repetition of g TFs

A schedule is constructed based on a sequence of choices as defined in Section 2.2.2: $S = \{..., C_{k'}^{j-1}, C_k^j, ...\}$. Therefore, vertices of a graph must be established to cover the choices. According to the definition of choice:

(1) Each vertex must be a set of g TFs chosen from the total number of K TFs
(2) Distinct TFs must be assigned within a set, i.e., the same TF can not appear twice in the set of any vertex.

From (1) and (2), it is induced that a vertex is a permutation without repetition of g TFs out of K TFs.

Observation

The order of TFs within a set vertex matters since, as discussed later, an edge between two vertices represents a compound forwarding from each of the TFs at the upstream node to each of TFs in the corresponding position at the downstream node, as exemplified in Figure 3.

Argument 2

If vertices at the first switch are combinations without repetition of g TFs, no redundant schedules are considered.

If vertices, at all switches, are constructed as permutations without repetition of g TFs out of K TFs, more than one resulting schedule might involve the same TFs at all switches on the path. Figure 3 shows an

example of schedule repetition where schedules S_1 and S_2 involve the same TFs at each switch.

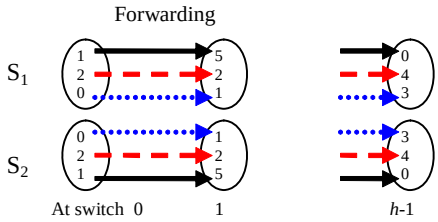

Figure 3 – Compound forwarding and schedule repetition example

Thus, computation complexity gets risk of being unnecessarily high due to this redundancy phenomenon. In order to solve the issue, vertices at the first switch are to be constructed as combination without repetition of *g* TFs out of *K* TFs. The solution stems from the fact that at the first switch, order of TF objects within a choice does not matter. The first switch is where forwarding starts, therefore there is no matter of forwarding from a previous TF to one of those TFs. Thus, from the two conditions (1) and (2) in Argument 1, vertices of the first switch can be constructed as combinations without repetition instead of permutations without repetition.

Moreover, due to the combination, no vertex of the first switch has the same set of TF objects. Thus, from the first switch to the second one, there is no same set of forwarding as descried above. Thus, no schedule is repeated over a route.

Vertices and graph construction

Based on the network model in 2.2.1, the forwarding graph is constructed as follows (see an example in Figure 4:

- Vertices of stage 0 are *combinations without repetition* of *g* TFs out of *K* TFs per TC. The total number of vertices is given: $N_0 = \frac{K!}{g!(K-g)!}$
- Vertices of the following (*h*-1) stages presents *permutations without repetition of g TFs*, representing a choice at the output link of switch *j*:

$$V_k^j = \left\{\left(TF_i^{\,j}\right)_l\right\}$$

Where

k: 0 → N_j , j: 0 → h-1,
i: 0 → K-1, l: 0 → g-1

The number of vertices at each of those (*h*-1) stages: $N_{j\neq 0} = \frac{K!}{(K-g)!}$

Graph component definitions:

Transition: a valid compound forwarding (defined in 2.2.2) between a pair of two available vertices of two sequential stages

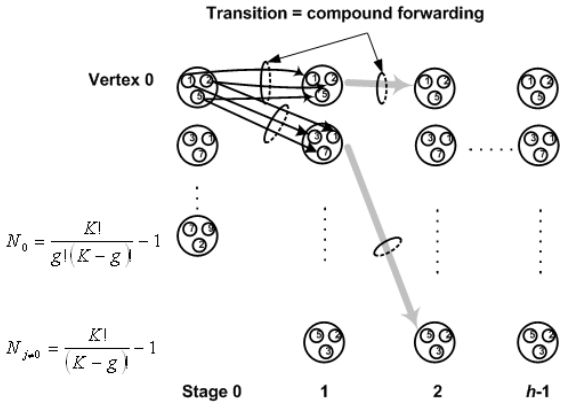

Figure 4 - A forwarding graph example

Transition metric: $\mu_b(k',k)$ - compound forwarding delay between 2 vertices of 2 consecutive stages.

A transition metric (transition delay) between vertices

$V_{k'}^{j-1} = \left\{\left(TF_{i'}^{\,j-1}\right)_l\right\}$ and $V_k^j = \left\{\left(TF_i^{\,j}\right)_l\right\}$

is determined by the maximum of all local metrics for each pair of local TFs:

$$\mu_b(k',k) = \max_{0\le l<g}\mu_l = \max_{0\le l<g}\left(\left(\left(TF_i^{\,j}\right)_l - \left(TF_{i'}^{\,j-1}\right)_l + K\right)\bmod K\right)$$

Path: a curve *P* from S to a vertex *k*-th on stage *j* - $P[k]$

Accumulated path metric: total sum of transition metric accumulated in a path *P* to the considered vertex: $\mu(P[k])$

Survivor: the smallest accumulated-metric path $\min(\mu(P[k]))$ among the ones incident to the considered vertex

Free vertex: a vertex is free if all TF objects within it are free.

3.2 Scheduling algorithm – eSSM

eSSM (**e**fficient **S**urvivor-based **S**earch for a **M**ultiple-TF session) algorithm searches for the smallest-delay schedule selection in a forwarding graph as described in the previous section. The scheduling idea is using ***Path filtration technique*** to compute stage by stage until the destination stage is reached.

The technique is elaborated as follows: given the path metric (accumulated delay) up to every available vertex of the stage (*j*-1), the searching algorithm works on each available vertex of stage *j* by comparing all path metrics to it, each consisting of the path metric built up to an available vertex of stage (*j*-1) plus the branch metric from it to the considered vertex of stage *j*.

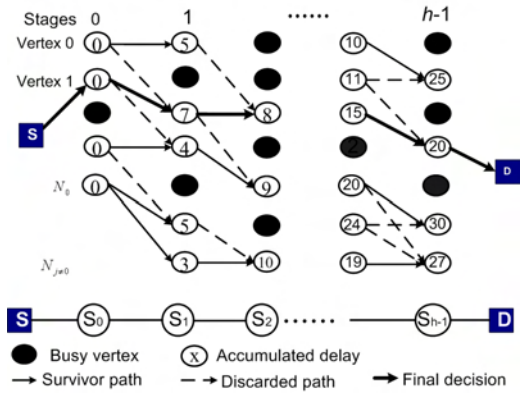

Figure 5 – eSSM example

For each available vertex of stage *j*, all those possible new paths coming to it are compared based on the overall path metric (i.e., the accumulated delay up to stage *j* -1 plus the forwarding delay corresponding to the selected branch) and only one best accumulated path, namely survivor, is kept for that vertex. This reflects the idea of searching for the smallest schedulable end-to-end forwarding delay, i.e., guaranteed quality of service. At the last stage, multiple survivor paths , for available vertices respectively, would be available. The last comparison now among those survivors yields **a final survivor schedule**.

Path Filtration

First of all, eSSM uses the function *Valid-Transition-Verification* to check if a transition between 2 vertices exists. Assume that:

$$V_{k'}^{j-1} = \left\{\left(TF_{i'}^{\,j-1}\right)_l\right\} \text{ and } V_k^j = \left\{\left(TF_i^{\,j}\right)_l\right\}$$

VALID-TRANSITION-VERIFICATION $(j, V_{k'}^{j-1}, V_k^j, Z)$

1. **If** vertices $V_{k'}^{j-1}$ and V_k^j are both free
2. **For** $l = 0{:}g\text{-}1$
3. **If** $\left[\left(TF_i^{\,j}\right)_l - \left(TF_{i'}^{\,j-1}\right)_l\right] \bmod K \le Z$
4. Transition $\left(V_{k'}^{j-1}, V_k^j\right) \leftarrow 1$

The algorithm then uses the technique of **Path Filtration**. At each stage $j \in$ transition graph, a minimized set of paths (a set of survivors) is selected by path filtration.

P is a set of survivors, each corresponds to a vertex *k'* at stage *j-1*, *P[k']=NIL* means vertex *k'* has no survivor (no scheduling possible via *k'*). *P'* is a newly computed survivor set at *j*. After all loops for each vertex, a set of survivors is determined for stage *j*, and ready for path filtration at stage *j*+1. Every path is a sequence of vertices on consecutive stages (one vertex at each stage) representing a flow schedule, with path metric $\mu(P[k])$ being the end-to-end delay of the flow.

PATH-FILTRATION (j, N_j)

1. **For** each free vertex $k \in$ stage j - V_k^j $(k: 0 \rightarrow N_j)$
2. $\mu_k = INF$
3. **For** each survivor $P[k'] \in P$
4. **If** transition $\left(V_{k'}^{j-1}, V_k^j\right) = 1$
5. $\mu_b(k', k) = \max_{0 \le l < g}\left(\left(\left(TF_i^{\,j}\right)_l - \left(TF_{i'}^{\,j-1}\right)_l + K\right) \bmod K\right)$
6. **If** $\mu(P[k']) + \mu_b(k', k) < \mu_k$
7. $\mu_k = \mu(P[k']) + \mu_b(k', k)$
8. $c = k'$
9. **Assign** $P'[k] \leftarrow \{P[c], k\}$

Step 1 and 3 show that only vertices which are free and reachable from previous vertices are considered in computation. Those vertices are called **involved vertices** as they are involved in algorithm computation. Considering only involved vertices help reducing the complexity as load increases.

Due to using the path filtration technique, eSSM inherits the optimality of eSS [7] in capable of yielding the same result as the exhaustive search does but with much lower complexity. eSSM can optimally cover all possibilities of finding all possibilities of assigning one TF or multiple TFs at each switch if a vertex presents a TF, or a group of multiple TFs respectively.

3.3 Complexity analysis

Algorithm complexity is defined as the number of computation iteration with which the algorithm has to go through, i.e. the number of transitions summing up from the ones incident to each vertex. Due to the Path Filtration technique, each previous vertex keeps only one path to advance further in eSSM. Thus, the transitions incident to each vertex is computed by counting the number of previous vertices that can transit to a current vertex.

3.3.1 Load = 0 or all K TFs are available

Note that due to the symmetry, the number of incident transitions is equal to the number of outgoing transitions for a vertex. Here we assess the lower- and upper- bound of the number of outgoing transitions of a vertex.

Lower bound: when distance between any 2 TFs of a vertex are assumed to be smaller than *Z*.

$$T_{LOW} = \frac{(Z+1)!}{(Z+1-g)!} \qquad \text{if } \forall d_i \le Z,\ g \le Z \quad (1)$$

Proof:

Each TF of a vertex has *Z*+1 single transitions. If the TFs are closer to each other then their transition spaces overlap more onto the other, leading to less compound transitions can be made. The lower bound happens when all *g* TFs of the vertex are continuous, and with

maximum overlap of single transitions among *g* TFs, i.e. when *Z*+1 > *g*.

We can derive the lower bound in that case by combining single transition possibilities:

- The 1*st* TF position of a vertex has (*Z*+1) possibilities of single transitions. The 2*nd* position has at least Z possibilities of TF to transit to after excluding the one for the 1st position.
- So on, the g-*th* TF position has at least [(*Z*+1)-g+1] possibilities

A minimum number of transitions to a vertex are given:

$$T_{LOW} = (Z+1)\cdot Z\cdot ...(Z+1-g+1) = \frac{(Z+1)!}{(Z+1-g)!}$$

Upper bound: when all distances between any 2 TFs of a vertex are assumed to be larger than the *Z*.

$$T_{UP} = (Z+1)^g \qquad \text{if } \forall d_i > Z \qquad (2)$$

Proof:

Thanks to the assumed condition, Z+1 single transitions of a TF do not overlap with any single transition of other TFs in that same vertex. Compound transitions are maximized as combinations of all g positions, each selected independently from *Z*+1 units. Thus, the maximum number of transitions for a vertex is:

$$T_{UP} = \underbrace{(Z+1)\cdot(Z+1)....(Z+1)}_{g} = (Z+1)^g$$

Complexity bounds*:* Since every of N_j vertices of switch *j>0* has the same upper and lower bound (1) (2) (for j=0, only one transition for each vertex), the complexity of eSSM over the route of *h* switches is given by:

$$X(h,K,Z,g) = \sum_{j=0}^{h-1}\sum_{i=0}^{N_j-1} T_i = N_0 + (h-1)\sum_{i=0}^{N_j-1} T_i \qquad (3)$$

Where

$$N_0 = \frac{K!}{g!(K-g)!};\ N_{j\neq 0} = \frac{K!}{(K-g)!}$$

$$T_i = (T_{LOW} : T_{UP})$$

From (3), it can be concluded that the complexity is linear in the size of *h*, and exponential in the size of *K* and *Z* –*O*(*h*,K^g,Z^g). Although the proposed algorithm is optimal solution in the sense of guaranteeing finding at least a schedule as long as a schedule exists (proved in [7]), the complexity is a considerable issue as the value *g* increases.

3.3.2 Load = B

In this section, the algorithm complexity is analyzed for a network load with B busy TFs per TC. The analysis is done by simulation of the eSSM, in which B busy TFs are uniformly distributed over each stage. Study of complexity in scenarios with load, especially high load is necessary due to its reality. Load helps ease the eSSM computation since the number of involved vertices in computation can be much less. Figure 6 illustrates a general changing behavior of complexity under load influence. The complexity is reduced in an exponential manner as load increases

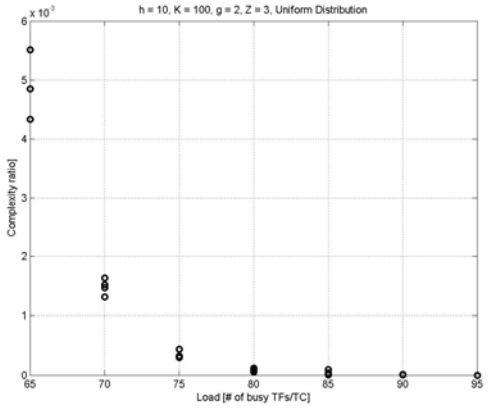

Figure 6 – Complexity ratio of different Loads vs. Load=0

4 Heuristic algorithm

4.1 Algorithm description

The behavior presented in Figure 6 gives us a hint that under a certain load threshold in a network scenario, a heuristic algorithm is efficient enough to replace eSSM. In fact, the eSSM is proved to be optimal in terms of guaranteeing finding a schedule if at least one schedule exists in the system. Therefore, eSSM should be used in the high load circumstance when finding a schedule is much more difficult because the existence of schedules is scare, while the complexity gets smaller. In case of low load, a heuristic algorithm is helpful with low complexity, while possibility of finding a schedule is high since resource is plentifully available.

We propose a heuristic algorithm that exploits the algorithm eSS [7] with the same core idea of using *Path filtration technique*. Unlike eSSM, eSS uses the simple forwarding graph with one TF per vertex to find single-TF flow each path. The simple forwarding graph includes *h* stages, each stage consisting *K* vertices standing for K TFs per TC. While at the end of eSS we select **one best path** from survivors of the last stage, now we need at best *g* survivors for the *g*-TF session. The problem is any pair of those *g* survivors must not contain any similar TF_i^j. Survivors without that overlapping are called unshared paths (*usp*), and a maximum set of *usp* is called S_{usp}. To retrieve unshared

paths, we observe a property from the transition graph and its important corollary:

PathShare property: *There are no two survivors joining to each other, but only splitting from a common path root from the first hop.*

Corollary: *The size of a S_{usp} equals to the number of root TFs (TF at first hop) presented in the last stage's survivor paths.*

The proof is easy considering the fact that eSSM keeps only one path per vertex on each stages, giving the survivor paths a tree-like model. As a result, it is not hard to build a function to check the overlapping vectors (paths) and to find a S_{usp} by browsing at the root TFs, which we implemented but will not present here for shortness of the paper.

The heuristic method can be described briefly as: run eSS, find S_{usp}, check if S_{usp} satisfies g-TF session, if not, add those S_{usp} to the schedule list, reserve (mark *busy*) all TFs of S_{usp} on the graph, and repeat from the beginning (run eSS with the new graph …) until the schedule list has enough g survivors.

This algorithm is heuristic in the sense that it considers the multi-TF requirement only at the last stage of eSS and if the unshared set of survivors is not enough for the g-TF, the eSS needs to run again with the smaller graph. Due to its sequential eSS execution, it does not ensure the optimality of scheduling multi-TF as in the eSSM. However, the heuristic is practically useful and good enough, since the real systems have large enough capacity compared to a session bandwidth, i.e. K >> g. When with low load, the chance of having large m and finding schedules is high, and the algorithm converges fast enough. After many sessions, when system gets busier, the eSSM can be efficiently applied as in 3.2

Heuristic(g,Z,K,h)

Initiate: number of remaining schedules: r = g

1. **Run** eSS(Z,K,h), find s survivors at the last stage;
2. **If** $s < r$
   P=Exit, failure: "not enough TF, use less or try later";
3. **If** $s \geq r$, **Identify a** S_{usp} *with* $m = size(S_{usp})$
4. **If** $m \geq r$
   $P \leftarrow$ (pick r best-delay paths from S_{usp});
   Exit, success. P is the schedule list
5. **If** $m < r$
   $P \leftarrow S_{usp}$; r = $r - m$;
6. **If** *size(P)* < g
   Mark TFs in S_{usp} as busy in the graph;
   Repeat from step 1

4.2 Complexity analysis

Its complexity, based on eSS, is as low as in the single TF schedule. The heuristic is quite efficient in terms of complexity compared to eSSM, with linear complexity in every size. Provided finding all g TFs needed for the session after v repetitions, the complexity of the heuristic is obviously v times as high as the complexity of the eSS [7]:

$$X(h,K,Z,g) = v\cdot(h\text{-}1)\cdot K\cdot(Z\text{+}1) \quad \text{where} \quad 1 \leq v \leq g$$

Figure 7 illustrates a simulation-based comparison between complexities of the heuristic and eSSM in a scenario: h=10, K=20, g=3, Z=10 and at least a schedule is found. The complexity of eSSM is 32 thousand times as high as the one of the heuristic at load 0% and ~ 230 times higher at load 65%.

Conclusively, for this scenario, load of ~70% can be a proper threshold to switch between two scheduling strategies: eSSM is used if load is higher than 70% and heuristic is used otherwise. This provides an experimental policy of load thresholds the network operator can use to apply suitable algorithms.

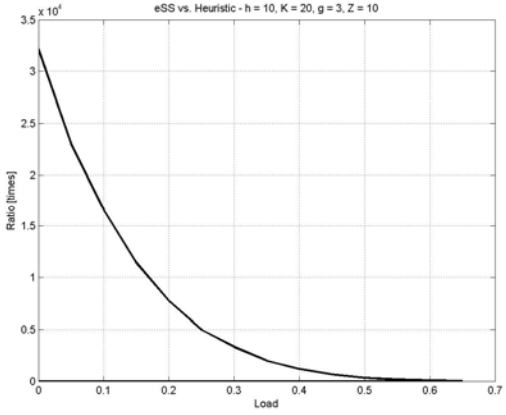

Figure 7 – complexity of eSSM vs. Heuristic

5 Scheduling policy towards packet order

Although IP networks do not assure that packets arrive to the destination in the same order in which they were sent, out-of-order packets can cause performance problems to many applications and higher layer protocols. Moreover, out-of-order packets cause additional delay in streaming media and interactive media applications, such as HDTV, Teleconferencing, Video-on-demand, Distance Learning, Telemedicine.

eSSM may decide a out-of-ordered schedule where TFs go out earlier from Source arrive later at Destination. It is because, with the graph and Path Filtration in eSSM, the algorithm is blind to the order requirement, and just selects schedules based on the vertex availability and delay optimizing. As far as we

have studied, the exhaustive search is the only algorithm that can evaluate all possibilities of in-order schedules at the expense of extremely high complexity. Thus, in order to trade off among criteria of appropriate complexity and efficient search for in-order delivery, we have eSSM modified in a way that it consistently searches over only ordered schedules. The solution is to run eSSM in a graph built up by only vertices that satisfy the increasing order of TFs. Thus, all choices of the schedule until the Destination have the same increasing order, as in the Source. The same eSSM idea can then perform on the new graph. Vertices are *involved* in the in-order graph as follows:

- V_k^0 are combinations of *g* out of *K* TFs, in which the *g* TFs are sorted in increasing order, e.g. $V_k^0 = (1,5,9)$
- V_k^h (h ≠ 0) for the in-order graph are selected from the vertices in eSSM graph if they contain TFs in increasing order within 2 time cycles. They are exactly the V_k^0 ones plus the cyclic-shift versions of V_k^0 ones, e.g. (2,3,1), (3,1,2).

6 Discussion

The exponential-like processing time with huge schedule space make scheduling for a multiple TF context a complex problem with many open issues. Our solution provides method to do that efficiently with some heuristics in practical scenarios. It is also a first step to further open the FλS network scheduling to various areas, e.g.: for a set of diverse session, for WDM (multiple channels), or with multicast, For all that, our eSSM application continues to looks possible with suitable modifications in future works.

References

[1] M. Baldi and Y. Ofek, "Fractional Lambda Switching - Principles of Operation and Performance Issues", *SIMULATION: Transactions of The Society for Modeling and Simulation International*, Vol. 80, No. 10, Oct. 2004, pp. 527-544.

[2] D. Grieco, A. Pattavina and Y. Ofek, "Fractional Lambda Switching for Flexible Bandwidth Provisioning in WDM Networks: Principles and Performance", *Photonic Network Communications*, Issue: Volume 9, Number 3, Date: May 2005, Pages: 281 – 296.

[3] V. T. Nguyen, et al, "Design and Analysis of Tunable Laser-based Fractional Lambda Switching," *IEEE INFOCOM 2006*.

[4] N.F. Huang, G.H Liaw, C.P Wang, "A Novel All Optical Transport Network with Time Shared Wavelength Channels", *IEEE Journals on selected areas in communications*, Vol.18, No.10, October 2000.

[5] Suresh Subramaniam, et al, "Scheduling Multirate Sessions in Time Division Multiplexed Wavelength Routing Networks", *IEEE Journal on Selected Areas in Communications, vol.18, no.10, October 2000*

[6] Bowen, Ramakrishna Shenai, and Krishna Sivalingam, "Routing, Wavelength and Time-Slot-Assignment Algorithms for Wavelength-Routed Optical WDM/TDM Networks", *Journals of Lightwave Technology*, Vol.23, No.9, September 2005.

[7] Thu-Huong Truong, Mario Baldi, Yoram Ofek, *"An Efficient Scheduling Algorithm for Time-Driven Switching Networks"*, IEEE LANMAN 2007