Optimal Scheduling of Two Communication Flows on Multiple Disjoint Packet-Type Aware Paths

Mugurel Ionut Andreica, Nicolae Tapus

Optimal Scheduling of Two Communication Flows on Multiple Disjoint Packet-Type Aware Paths

Mugurel Ionuţ Andreica, Nicolae Ţăpuş
Computer Science and Engineering Department
Politehnica University of Bucharest
Bucharest, Romania
{mugurel.andreica, nicolae.tapus}@cs.pub.ro

Abstract—**Communication flows in distributed systems often present a poor performance, because they are unaware of each other and end up competing for the same bottleneck resources. A solution to this problem consists of scheduling the communication flows in order to optimize some performance metric. In this paper we study the scheduling of two communication flows over multiple disjoint paths, such that the maximum completion time (makespan) is minimized. Each flow is composed of a large number of identical packets of the same type. The paths are aware of the packet types and have different transmission times for each type. We consider the objectives of minimizing the makespan and the weighted sum of completion times. We also consider some error-correcting issues, as well as the possibility of dropping the packet ordering constraints.**

Keywords-communication flows; scheduling; disjoint paths

I. INTRODUCTION

Communication performance in distributed systems may be quite poor when multiple communication flows use the network simultaneously. Because they are not aware of each other, they end up trying to use the same bottleneck resources, although other resources may be available in other places or at other times. A solution to this problem consists of scheduling the communication flows in such a way that a performance metric is optimized. In this paper we are interested in optimally scheduling two communication flows from the same sender to the same receiver, using multiple disjoint paths. Each flow i is composed of a number of identical packets which need to be sent sequentially. The paths have different transmission times for the two packet types. This kind of situation may occur when the two communication flows belong to distinct traffic classes (for instance, multimedia and normal web traffic). Moreover, some paths may be more suitable for one of the two traffic types. We show that, when the objective is to minimize the makespan, optimal schedules present very particular structures, considering that the packet transmission is non-preemptive and that two packets cannot be in transit at the same time on the same path. We also consider the objective of minimizing the sum of completion times and present dynamic programming algorithms for two situations.

The rest of this paper is structured as follows. In Section II we define the makespan minimization problem. In Section III we characterize the structure of minimum makespan schedules and in Section IV we present an algorithm for computing an optimal schedule. In Section V we consider the problem of minimizing the sum of completion times. In Section VI we drop the packet ordering constraints and in Section VII we consider error-correcting issues. In Section VIII we present related work and in Section IX we conclude.

II. MINIMUM MAKESPAN SCHEDULING

We consider *2* communication flows, composed of $np(i)$ identical packets ($i=1,2$). The packets of the communication flow i are of type i and are identified by a pair of numbers (i,j), $1 \le j \le np(i)$. The packets of the same type must be sent to the destination sequentially, using P disjoint paths. Each path q ($1 \le q \le P$) has *2* transmission times, $ts(q,1)$ and $ts(q,2)$. $ts(q, i)$ ($i=1,2$) is the time taken by a packet of type i to reach the destination using path q. A schedule consists of assigning to each packet (i,j) a pair $(path(i,j), tstart(i,j))$, meaning that the packet is scheduled to be sent on path $path(i,j)$, starting from time $tstart(i,j)$. Based on this pair, we also associate with each packet (i,j) a time interval $[tstart(i,j), tfinish(i,j))$, where $tfinish(i,j)=tstart(i,j)+ ts(path(i,j),i)$. During this interval, the packet (i,j) is in transit on $path(i,j)$, so we will call it *transit interval*. A schedule is valid if for any two packets $(i,j(1))$ and $(i,j(2))$, $j(1)<j(2)$, we have $tfinish(i,j(1)) \le tstart(i,j(2))$, and if any two packets scheduled on the same path (disregarding their type) are assigned disjoint transit intervals. The first condition makes sure that each flow's packets are sent sequentially (in the order given by the starting time of the packets' transit intervals) and the second one makes sure that the packets scheduled on the same path are sent one at a time. The makespan C_{max} of the schedule is the maximum time at which a packet's transmission ends and we want to find a schedule which minimizes C_{max}:

$$C_{\max} = \max_{i=1,2}\{tfinish(i, np(i))\}. \qquad (1)$$

III. PROPERTIES OF OPTIMAL SCHEDULES

In this section we show that an optimal schedule (which minimizes the makespan) must have a particular structure, chosen from a small set of such structures. As a first step, we show that in an optimal schedule, each flow's packets are scheduled on at most *3* distinct paths. In order to do this, we will present and prove several theorems. The main technique lying at the basis of all the proofs is choosing an arbitrary valid schedule and changing it into a schedule which is not worse, but has all the properties mentioned by the theorem. For each flow *i*, we define an ordering of the paths: *po(i,1), po(i,2), …, po(i,P)*, such that $ts(po(i,1),i) \leq ts(po(i,2),i) \leq \ldots \leq ts(po(i,P),i)$. In the proofs of the following theorems we will frequently reassign a packet from a path *po(i,q(1))* to a path *po(i,q(2))*, with $q(2)<q(1)$. Such a reassignment does not change the starting time of the packet, but may decrease its ending time. The makespan of the schedule will not increase as a result of these operations.

Theorem 1. *Let k be the first position where the path orderings of the two flows differ, i.e. $po(1,q)=po(2,q)$, for $1 \leq q<k$ and $po(1,k) \neq po(2,k)$. If such a position exists, then in an optimal schedule, no packets are sent on any of the paths po(i,q) (i=1,2), with $q>k$.*

Proof. We will choose an arbitrary valid schedule. All the packets *(i,j)* which are assigned to paths *path(i,j)* such that $path(i,j)=po(i,q)$, $q>k$, will be reassigned to path *po(i,k)*. After this reassignment, we obtain a new schedule. We will analyze the validity of this new schedule. The reassignment does not change the starting time of any packet, only the finish time, which may decrease. Therefore, the transit intervals of packets of the same type do not overlap. Let's see if the transit intervals of two packets scheduled on the same path might intersect. If the two packets are of the same type, we showed previously that this cannot happen, because their transit intervals are disjoint. Let's assume that the transit intervals of two packets of different types, *(1,j(1))* and *(2,j(2))*, scheduled on the same path *p*, intersect. This path cannot be one of the first *k-1* paths (for any of the two flows), because no packet was reassigned to such a path. So path *p* must be the k^{th} path of one of the flows. W.l.o.g., we will assume that *p=po(1,k)*. But no type *2* packet is assigned to path *po(1,k)*, thus invalidating our initial assumption. In conclusion, the new schedule is valid and this holds for any valid schedule, including the optimal one.

Theorem 2. *In an optimal schedule, no packet (i,j) is sent on a path po(i,q), with $q>4$.*

Proof. We will choose an arbitrary valid schedule, where at least one packet *(i(1),j)* is scheduled on a path *po(i(1),q)*, with *q>4* (using Theorem *1*, we also have $q \leq k$, if *k* exists). The packets of the other type *i(2)* can be classified into *3* categories, according to the relationship between their transit interval and packet *(i(1),j)*'s transit interval:

- category *1*: their transit interval is included inside *[tstart(i(1),j), tfinish(i(1),j))*.
- category *2*: their transit interval intersects *[tstart(i(1),j), tfinish(i(1),j))*, but is not included in it.
- category *3*: their transit interval does not intersect *[tstart(i(1),j), tfinish(i(1),j))*.

All the category *1* packets can be reassigned to path *po(i(2),1)*, because no other packet of type *i(1)* is scheduled on that path during the interval *[tstart(i(1),j), tfinish(i(1),j))*. The transit intervals of these packets do not increase. Category *2* packets cannot be reassigned to a different path, because they might be in conflict with other type *i(1)* packets. However, it is easy to see that there can be at most two packets belonging to category *2*. One of the packets may cross *[tstart(i(1),j), tfinish(i(1),j))* at the left end and the other one at the right end. Category *3* packets are of no interest. After performing the reassignment of category *1* packets, the packet *(i(1),j)*'s transit interval intersects with the transit intervals of packets scheduled on at most three distinct paths. So the packet *(i(1),j)* can be reassigned to at least one of the paths *po(i(1),1)*, …, *po(i(1),4)*, without increasing the makespan and without breaking the validity of the schedule. By using this reassignment method repeatedly, all the packets *(i(1),j)* assigned to some path *po(i(1),q)*, *q>4*, will be reassigned to some path *po(i(1), q(1))*, $1 \leq q(1) \leq 4$. Thus, any valid schedule can be turned into another schedule, where no packet is assigned to a path *po(i,q)*, *q>4*. This holds for an optimal schedule, too.

Theorem 3. *In an optimal schedule, no packet (i,j) is sent on a path po(i,q), with $q>3$.*

Proof. We will choose an arbitrary valid schedule. We first use Theorem *2* in order to obtain a valid schedule with a makespan which is not worse that the initial schedule and in which no packet is sent on a path *po(i,q)*, with *q>4*. Let's assume that a packet *(i(1),j)* is assigned to the path *po(i(1), 4)*. From Theorem *1*, we must have $k \geq 4$. We will classify the type $i(2)(\neq i(1))$ packets in the same three categories as in Theorem *2*'s proof and perform the same reassignments. If there are less than two packets in category *2* or if the two packets in category *2* are assigned to the same path or if one of them is assigned to path *po(i(2),1)* or to path *po(i(2),4)* then packet *(i(1),j)*'s transit interval intersects the transit intervals of packets scheduled on at most two distinct paths from the set *{po(i(1),1), po(i(1),2), po(i(1),3)}*, which allows us to reassign packet *(i(1),j)* to one of the paths in the set. The more difficult case occurs when there are two packets belonging to category *2*, one of them assigned to the path *po(i(2),2)* and the other one to the path *po(i(2),3)* (see Fig. 1). In Fig. 1, w.l.o.g., we chose to place the type *i(2)* packet assigned to path *po(i(2),2)* on the left. Because each flow's schedule begins at time *0* and ends at time C_{max}, we can always choose to interpret time as moving from C_{max} towards *0*, so left and right are interchangeable. We will name *(i(2),j(2))* and *(i(2),j(3))* the two type *i(2)* packets assigned to paths *po(i(2),2)* and *po(i(2),3)*. All the type *i(1)* packets whose transit intervals start after the finish time of packet *(i(1),j)* and finish before the finish time of *(i(2),j(3))*, or finish before the starting time of *(i(1),j)* and start after the starting time of *(i(2),j(2))* can be reassigned to path *po(i(1), 1)*. Packet *(i(2),j(3))*'s transit interval must intersect with that of a packet of type *i(1)* assigned to path *po(i(1),2)*; otherwise, packet *(i(2),j(3))* could

be reassigned to path *po(i(2),2)* and then packet *(i(1),j)* could be reassigned to path *po(i(1),3)*. We define the interval $[t_1,t_2)$, where t_1 is the starting time of the first type *i(1)* packet (re)assigned to path *po(i(1),1)* whose transit interval is fully included inside that of packet *(i(2),j(2))* (or, if no such interval exists, the starting time of packet *(i(1),j))* and t_2 is the starting time of the first type *i(1)* packet assigned to *po(i(1),2)* whose transit interval intersects that of the packet *(i(2),j(3))*. We also define t_3 as the finish time of the transit interval of packet *(i(2), j(3))*. We define *l(i(1),1)* the total length of the transit intervals included in $[t_1,t_2)$ of all the type *i(1)* packets

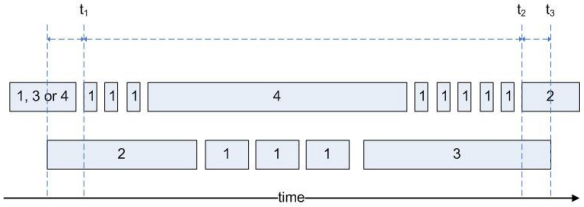

Figure 1. Type *i(1)* packets (first row) and type *i(2)* packets (second row). The left and right side of the packets are aligned with their starting and finish time. The numbers inside the packets are the positions of the assigned paths in the corresponding path ordering.

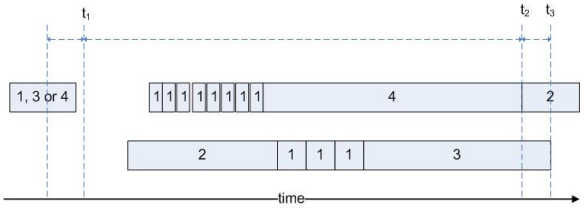

Figure 2. The case $l(i(2),3)+l(i(2),1) \leq t_3-t_2+l(i(1),4)$. The rearrangement of packets. No path reassignment has been performed, yet.

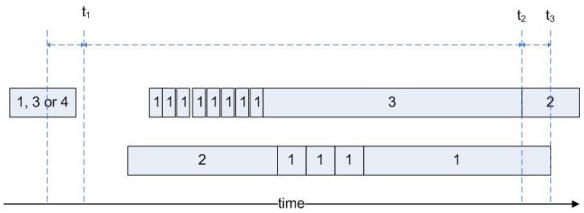

Figure 3. The new schedule in the case $l(i(2),3)+l(i(2),1) \leq t_3-t_2+l(i(1),4)$. The schedule is valid and the makespan did not increase.

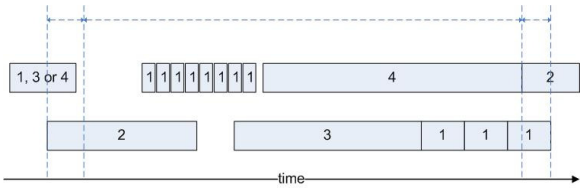

Figure 4. The case $l(i(2),1)+l(i(2),3) > t_3-t_2+l(i(1),4)$. The rearrangement of packets. No path reassignment has been performed, yet.

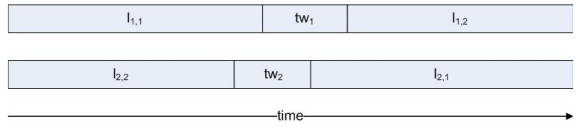

Figure 5. Rearrangement of packets for the case k=2, P≥2.

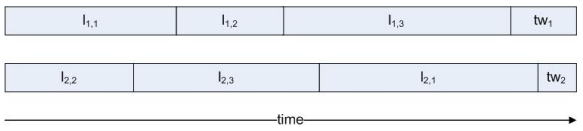

Figure 6. The case k=3, P≥4. Packet rearrangement when $l_{1,1}+l_{1,2} \leq l_{2,2}+l_{2,3}$. No path reassignment has been performed, yet.

(re)assigned to path *po(i(1),1)* and *l(i(1),4)* the length of the transit interval of the packet *(i(1),j)*. Similarly, we define *l(i(2),1)* the total length of the transit intervals included in $[t_1,t_2)$ of all the type *i(2)* packets (re)assigned to path *po(i(2),1)*, *l(i(2),2)* the

length of the transit interval of the packet *(i(2),j(2))* and *l(i(2),3)* the length of the transit interval of the packet *(i(2),j(3))*. All the packets whose transit intervals are included inside $[t_1,t_2)$ will be rearranged in such a way that the makespan will not increase and that it will be possible to reassign packet *(i(1),j)* to one of the paths *po(i(1),1),…,po(i(1),3)*.

If $l(i(2),3)+l(i(2),1)\le t_3-t_2+l(i(1),4)$, then the packets can be rearranged like in Fig. 2. Packet *(i(1),j)* is placed such that its finish time is equal to t_2. Then, all the other type *i(1)* packets whose transit intervals were included in $[t_1,t_2)$ will be placed somewhere inside the interval $[t_1,t_2-l(i(1),4))$. This is obviously possible, because $l(i(1),1)\le t_2-t_1-l(i(1),4)$. After that, the packet *(i(2),j(3))* will be reassigned to path *po(i(2), 1)*. This is now possible, because the only two packets whose transit intervals intersect the transit interval of packet *(i(2),j(3))* are assigned to the paths *po(i(1),2)* and *po(i(1),4)*. After packet *(i(2),j(3))* is reassigned to path *po(i(1),1)*, packet *(i(1),j)* can be reassigned to path *po(i(1),3)*. The final arrangement is shown in Fig. 3.

The case $l(i(2),3)+l(i(2),1)>t_3-t_2+l(i(1),4)$ is handled in a similar manner. The type *i(1)* packets whose transit interval is included inside $[t_1,t_2)$ are placed like in the previous case. The type *i(2)* packets (re)assigned to path *po(i(1),1)* whose transit intervals are included inside $[t_1,t_2)$ are rescheduled consecutively, such that the last one finishes at time t_3. Because $l(i(2),1)<l(i(1),4)$ (initially, all of these packets' intervals were included inside packet *(i(1),j)*'s transit interval), the new starting time of the first of these packets, $t_{newstart}=t_3-l(i(2),1)$, will be greater than the new starting time of the packet *(i(1),j)*. The packet *(i(2),j(3))* will be rescheduled such that its finish time is equal to $t_{newstart}$. Because $l(i(2),1)+l(i(2),3)>t_3-t_2+l(i(1),4)$, the starting time of the packet *(i(2),j(3))* will be smaller than the starting time of packet *(i(1),j)*. Fig. 4 shows the new arrangement. The new transit interval of packet *(i(1),j)* intersects only type *i(2)* packets assigned to the paths *po(i(2),1)* and *po(i(2),3)*, so packet *(i(1),j)* can be reassigned to path *po(i(1),2)*. This was the last case to be considered. In every case, packet *(i(1),j)* could be reassigned to a path *po(i(1),q(1))*, with $1\le q(1)\le 3$, without assigning any packet to a path located on a larger position in the corresponding path ordering and without increasing the makespan. Thus, any valid schedule (including an optimal one) can be changed into another valid schedule where no packet is assigned to a path *po(i,q)*, $q>3$.

We will characterize next all the cases of interest that may occur, according to the total number of paths P and the parameter k defined in Theorem *1*. We will use *A div B* to denote the integer division of A and B, i.e. the integer number C, such that $C\cdot B\le A<(C+1)\cdot B$. A and B do not necessarily have to be integer numbers.

A. k=1, P≥1

If *po(1,1)*$\neq$*po(2,1)*, then all the type *1* packets will be scheduled on path *po(1,1)* and all the type *2* packets on the path *po(2,1)*. The makespan will be:

$$C_{\max}=\max\{np(1)\cdot ts(po(1,1),1), np(2)\cdot ts(po(2,1),2)\}. \quad (2)$$

B. k does not exist, P=1

If *P=1* and *po(1,1)=po(2,1)*, then all the packets of both types will be scheduled on the first (and only) path. The makespan will be

$$C_{\max}=np(1)\cdot ts(1,1)+np(2)\cdot ts(1,2). \quad (3)$$

C. k=2, P≥2

We choose an arbitrary valid schedule and denote its makespan by C. We denote by $l_{i,j}$ the total length of the transit intervals of type i packets scheduled on path *po(i,j)* ($1\le j\le 2$) and by tw_i the total waiting time $tw_i=C-l_{i,1}-l_{i,2}$.

We have that $l_{1,1}\le l_{2,2}+tw_2$, because each transit interval of a type *1* packet scheduled on the path *po(1,1)* overlaps some part of a type *2* packet scheduled on path *po(2,2)* or some part of the waiting time tw_2. Because of this, the schedule can be changed such that all the type *1* packets assigned to path *po(1,1)* are scheduled first, followed by the waiting time tw_1 and then by all the type *1* packets scheduled on path *po(1,2)*. For the 2^{nd} flow, all the packets assigned to path *po(2,2)* are scheduled first, followed by the waiting time tw_2 and by the packets assigned to path *po(2,1)*. Fig. 5 presents the transformed schedule.

No transit interval of a type *1* packet scheduled on path *po(1,1)* overlaps with the transit interval of a type *2* packet scheduled on path *po(2,1)*. The schedule can be further refined by moving part of the waiting time tw_2 at the end and moving the type *2* packets assigned to path *po(2,1)* forward, so that their starting time is $\max\{l_{1,1}, l_{2,2}\}$. Similarly, part of tw_1 can be moved at the end, so that type *1* packets assigned to path *po(1,2)* are sent starting from $\max\{l_{1,1}, l_{2,2}\}$. Obviously, the new schedule is valid and its makespan is not larger than that of the original schedule.

An optimal schedule is properly defined by the number u of type *1* packets assigned to path *po(1,1)*. For the type *1* packets, the schedule can be written as $1^u,2^{np(1)-u}$, meaning that the first u packets are assigned to path *po(1,1)* and the last *np(1)-u* packets are assigned to path *po(1,2)* (a term of the form a^b in the schedule of flow i represents b consecutive type i packets sent on path *po(i,a)*). If the number u of packets is fixed, the schedule for the type *2* packets has one of the following two forms:

- $2^v,tw,1^{np(2)-v}$, where $v=\min\{(l_{1,1} \text{ div } ts(po(2,2),2)), np(2)\}$ and $tw=l_{1,1}-v\cdot ts(po(2,2),2)$. This means that the first v packets of type *2* are sent consecutively on the path *po(2,2)*, then a waiting time tw follows and then the last *np(2)-v* type *2* packets are sent on the path *po(2,1)*.
- $2^v,1^{np(2)-v}$, where $v=\min\{np(2),(l_{1,1} \text{ div } ts(po(2,2),2))+1\}$

In order to find the optimal schedule, we need to find the value of u which minimizes the makespan.

D. k does not exist, P=2

This case is similar to the previous one. We will use the same notations as before. We have that $l_{1,1} \leq l_{2,2}+tw_2$ and $l_{2,2} \leq l_{1,1}+tw_1$ (by the same argument). Therefore, the schedule shown in Fig. 5 is valid in this case, too. Like in the previous case, an optimal schedule is properly defined by the number u of type *1* packets assigned to path *po(1,1)*. These packets will be sent first. In parallel, we will send as many type *2* packets as possible on path *po(2,2)*; we have two choices:

- send $v=min\{(l_{1,1} \text{ div } ts(po(2,2),2)),np(2)\}$ type *2* packets on path *po(2,2)*, then wait a duration $tw=l_{1,1}-v \cdot ts(po(2,2),2)$ and then send the remaining type *2* packets on path *po(2,1)*. The type *1* packets assigned to path *po(1,2)* will be sent starting from time $l_{1,1}$. The schedule for the type *1* packets is $1^u,2^{np(1)-u}$ and the schedule for type *2* packets is $2^v,tw,1^{np(2)-v}$.
- send $v=min\{(l_{1,1} \text{ div } ts(po(2,2),2))+1,np(2)\}$ type *2* packets on path *po(2,2)*, then send immediately the remaining type *2* packets on path *po(2,1)*. The type *1* packets assigned to path *po(1,2)* will have to wait a duration $tw=max\{v \cdot ts(po(2,2),2)-l_{1,1},0\}$ before starting to send them. The schedule for the type *1* packets is $1^u,tw,2^{np(1)-u}$ and the one for type *2* packets is $2^v,1^{np(2)-v}$. In this case, it would be better to choose the value of v and derive the value of u based on v.

Like in the previous case, finding the optimal schedule means finding the value of u which minimizes the makespan.

E. k=3, P≥4

If no packet *(i,j)* is assigned to the path *po(i,3)*, then this case is identical to the previous one. So we will restrict our attention to the case in which at least one packet *(i,j)* is assigned to the path *po(i,3)*. We will choose an arbitrary valid schedule with makespan *C*. We will define $l_{1,1}$, $l_{1,2}$, $l_{2,1}$, $l_{2,2}$ as before. Furthermore, we define $l_{i,3}$ the total length of the transit intervals of the type i packets assigned to path *po(i,3)* ($1 \leq i \leq 2$). The waiting times are now equal to $tw_i=C-l_{i,1}-l_{i,2}-l_{i,3}$. If $l_{1,1}+l_{1,2} \leq l_{2,2}+l_{2,3}$, the packets can be rearranged like in Fig. 6 (temporarily, packets of both types sent on the path *po(1,2)=po(2,2)* may intersect). All type *1* packets assigned to path *po(1,1)* will be sent first, followed by all the type *1* packets assigned to path *po(1,2)* and by all the type *1* packets assigned to path *po(1,3)*. In parallel, we will send all the type *2* packets assigned to path *po(2,2)*, followed by all the type *2* packets assigned to path *po(2,3)* and then followed by those assigned to path *po(2,1)*. The waiting times are moved at the end of the schedule.

The type *1* packets assigned to path *po(1,2)* will be reassigned to path *po(1,1)*. The type *2* packets assigned to path *po(2,3)* will be reassigned to path *po(2,2)*. At this point, the type *1* packets are assigned only to the paths *po(1,1)* and *po(1,3)* and the type *2* packets are assigned only to the paths *po(2,1)* and *po(2,2)*. However, more reassignments are possible. All type *1* packets assigned to path *po(1,3)* whose finish time is smaller than or equal to $l_{2,2}+l_{2,3}$ can be reassigned to path *po(1,1)*. All type *1* packets assigned to path *po(1,3)* whose starting time is greater than or equal to $l_{2,2}+l_{2,3}$ can be reassigned to path *po(1,2)*. All these reassignments do not increase the lengths of the transit intervals, so they do not increase the makespan. In the end, there will be at most one type *1* packet assigned to path *po(1,3)* and no type *2* packet assigned to path *po(2,3)*. The schedule for the type *1* packets has the form $1^u,3^1,2^{np(1)-u-1}$ and the one for type *2* packets has the form $2^v,1^{np(2)-v}$.

If $l_{1,1}+l_{1,2}>l_{2,2}+l_{2,3}$ and $l_{1,1} \geq l_{2,2}+l_{2,3}$, we can change the schedule in a similar manner. We will send the first type *1* packets assigned to path *po(1,1)*, followed by the type *1* packets assigned to path *po(1,2)* and then *po(1,3)*. In parallel, the type *2* packets assigned to path *po(2,2)* will be sent, followed immediately by the packets assigned to path *po(2,3)*. Because we have $l_{1,1} \leq l_{2,2}+l_{2,3}+tw_2$ (since any transit interval of a type *1* packet assigned to path *po(1,1)* overlaps parts of transit intervals of type *2* packets assigned to paths *po(2,2)* or *po(2,3)*, or parts of tw_2), we can insert the waiting time tw_2 before sending the type *2* packets assigned to path *po(2,1)*. This way, the makespan does not increase and the schedule remains valid. Further reassignments are possible. All type *2* packets assigned to path *po(2,3)* will be reassigned to path *po(2,2)* and all type *1* packets assigned to path *po(1,3)* will be reassigned to path *po(1,2)*. This way, no packet *(i,j)* remains assigned to the path *po(i,3)*, so we are in the case presented in the previous subsection. If $l_{1,1}+l_{1,2}>l_{2,2}+l_{2,3}$ and $l_{1,1}<l_{2,2}+l_{2,3}$, we need to make a difference between the following subcases: $l_{1,1} \geq l_{2,2}$ and $l_{1,1}<l_{2,2}$.

Subcase *1*: $l_{1,1} \geq l_{2,2}$. The packets will be rearranged the same way as before: for type *1* - the packets assigned to the path *po(1,1)*, then those assigned to path *po(1,2)* and then those assigned to path *po(1,3)*; for type *2* - the packets assigned to path *po(2,2)*, then those assigned to path *po(2,3)* and then those assigned to path *po(2,1)*. Because $l_{1,1} \geq l_{2,2}$, the transit interval of no type *1* packet assigned to paths *po(1,2)* or *po(1,3)* overlaps the transit interval of a type *2* packet assigned to path *po(2,2)*. Thus, all the type *1* packets assigned to path *po(1,3)* can be reassigned to path *po(1,2)* and the schedule is valid. The type *2* packets assigned to path *po(2,3)* whose finish time is smaller than or equal to $l_{1,1}$ will be reassigned to path *po(2,2)* and those whose starting time is greater than or equal to $l_{1,1}$, will be reassigned to path *po(2,1)*. This leaves at most one type *2* packet still assigned to path *po(2,3)*. The schedule for the type *1* packets has the form $1^u,2^{np(1)-u}$ and the one for type *2* packets has the form $2^v,3^1,1^{np(2)-v-1}$.

Subcase 2: $l_{1,1}<l_{2,2}$. The type *2* packets will be rearranged just like in the previous subcase. Furthermore, all the type *2* packets assigned to path *po(2,3)* will be reassigned to path *po(2,1)*. The type *1* packets assigned to path *po(1,1)* will be sent first, followed by the type *1* packets assigned to path *po(1,3)*. Because $l_{1,1}+l_{1,3}+tw_1 \geq l_{2,2}$, we can insert the waiting time tw_1 before sending the type *1* packets assigned to path *po(1,2)*. After the reassignments, the schedule is valid and its makespan did not increase. Furthermore, the type *1* packets assigned to path *po(1,3)* whose finish time is smaller than or equal to $l_{2,2}$ will be reassigned to path *po(1,1)* and those whose starting time is greater than or equal to $l_{2,2}$ will be reassigned to path *po(1,2)*,

leaving at most one type *1* packet still assigned to path *po(1,3)*. The schedule for the type *1* packets has the form $1^u,3^1,2^{np(1)-u}$ or $1^u,tw,2^{np(1)-u}$ and the one for type *2* packets has the form $2^v,1^{np(2)-v}$.

F. *k does not exist, P=3*

Any valid schedule for this case is also a valid schedule for the previous one. Therefore, we can use the same arguments and transformations. The only problem we might encounter is that the schedule obtained after performing the transformations of the previous case might contain two packets *(1,j(1))* and *(2,j(2))*, with overlapping transit intervals and assigned to the same path *po(1,3)=po(2,3)*. However, we can see that this is not the case, because any schedule obtained in the previous case contains at most one packet *(i,j)* assigned to a path *po(i,3)* (either *po(1,3)* or *po(2,3)*).

G. *k>3 or non-existent, P>3*

According to Theorem *3*, no packet *(i,j)* is sent on a path *po(i,q)*, *q>3*. Thus, we can limit the value of *P* to *3* and the case becomes identical to the previous one.

In this section we characterized the structure of optimal schedules. There are five kinds of non-trivial structures:

- $1^u,2^{np(1)-u}$ for flow *1* and $2^v,tw,1^{np(2)-v}$ for flow *2*, where $v=min\{np(2), ((ts(po(1,1),1)\cdot u)\ div\ ts(po(2,2),2))\}$ and $tw=u\cdot ts(po(1,1),1)-v\cdot ts(po(2,2),2)$.
- $1^u,tw,2^{np(1)-u}$ for flow *1* and $2^v,1^{np(2)-v}$ for flow *2*, where $u=min\{np(1), ((ts(po(2,2),2)\cdot v)\ div\ ts(po(1,1),1))\}$ and $tw=v\cdot ts(po(2,2),2)-u\cdot ts(po(1,1),1)$.
- $1^u,2^{np(1)-u}$ for flow *1* and $2^{v+1},1^{np(2)-v-1}$ for flow *2*, where $v=min\{np(2)-1, ((ts(po(1,1),1)\cdot u)\ div\ ts(po(2,2),2))\}$.
- $1^u,2^{np(1)-u}$ for flow *1* and $2^v,3^1,1^{np(2)-v-1}$ for flow *2*, where $v=min\{np(2)-1,((ts(po(1,1),1)\cdot u)\ div\ ts(po(2,2),2))\}$.
- $1^u,3^1,2^{np(1)-u-1}$ for flow *1* and $2^v,1^{np(2)-v}$ for flow *2*, where $u=min\{np(1)-1,((ts(po(2,2),2)\cdot v)\ div\ ts(po(1,1),1))\}$.

IV. A Makespan Minimization Algorithm

We will present an algorithm with time complexity *O(np(i))* which determines the optimal schedule for any of the five kinds of non-trivial structures presented in the previous section. The algorithm has time complexity *O(log(np(i)))* on three of the schedule structures, but two structures are more difficult and we were unable to develop an equally efficient algorithm for them. We will not include in this section the trivial cases *k=1* and *P=1*, which can easily be solved in *O(1)* time using equations (2) and (3).

A. *Case 1: $1^u,2^{np(1)-u}$ and $2^v,3^1,1^{np(2)-v-1}$*

We chose to handle first the case $1^u,2^{np(1)-u}$ and $2^v,3^1,1^{np(2)-v-1}$, because it is easier to solve than the cases where waiting times are involved. We will define two functions, $C_1(u)$ and $C_2(u)$ representing the completion time of flow *1* and flow *2*, respectively, if there are *u* packets of type *1* assigned to the path *po(1,1)*. Their definitions are:

$$C_1(u)=u\cdot ts(po(1,1),1)+(np(1)-u)\cdot ts(po(1,2),1) \quad (4)$$

$$C_2(u)=v\cdot ts(po(2,2),2)+ts(po(2,3),2)+(np(2)-v-1)\cdot ts(po(2,1),2) \quad (5)$$

, with $v=min\{(ts(po(1,1),1)\cdot u)\ div\ ts(po(2,2),2)\ ,\ np(2)-1\}$

The first function is decreasing for $u\in[0,np(1)]$. The difference $C_1(x+1)-C_1(x)=ts(po(1,1),1)-ts(po(1,2),1)$ is constant. The values of the second function are increasing, but not necessarily strictly increasing. This is easily noticeable, because as *u* increases, so does *v*. Whenever *v* increases, the number of packets assigned to path *po(2,2)* increases and the number of packets assigned to path *po(2,1)* decreases, so the overall value increases. In order to find the value $u_{opt}\in[0,np(1)]$ for which $max\{C_1(u_{opt}), C_2(u_{opt})\}$ is minimum we have the following three subcases:

- Subcase 1: $C_1(0)\le C_2(0)$. In this case, $max\{C_1(u), C_2(u)\}=C_2(u)$ and the minimum value of $C_2(u)$ is $C_2(0)$. So $u_{opt}=0$.
- Subcase 2: $C_1(np(1))\ge C_2(np(1))$. In this case, $max\{C_1(u), C_2(u)\}=C_1(u)$ and the minimum value of $C_1(u)$ is $C_1(np(1))$. So $u_{opt}=np(1)$.
- Subcase 3: $C_1(w)\ge C_2(w)$, for $0\le w\le w_m$ and $C_1(w)<C_2(w)$ for $w_m<w\le np(1)$. We can find the value of w_m using binary search. The value of u_{opt} is either w_m or w_m+1.

The time complexity of the algorithm is $O(log(np_i))$. The cases $((1^u,3^1,2^{np(1)-u-1}), (2^v,1^{np(2)-v}))$ and $((1^u,2^{np(1)-u}), (2^{v+1},1^{np(2)-v-1}))$ are similar. We define the two functions $C_1(v)$ and $C_2(v)$ having the same meaning, which are decreasing, respectively strictly increasing. We have the same three situations and use binary search to find the optimal value u_{opt} in the third situation.

B. *Case 2: $1^u,2^{np(1)-u}$ and $2^v,tw,1^{np(2)-v}$*

We will define the two functions $C_1(u)$ and $C_2(u)$, representing the completion time of the first, respectively, second flow, if *u* packets of type *1* are assigned to path *po(1,1)*. C_1 is defined as before, while C_2's definition is:

$$C_2(u)=u\cdot ts(po(1,1),1)+(np(2)-v)\cdot ts(po(2,1),2)\ ,\ \text{with} \quad (6)$$

$v=min\{(ts(po(1,1),1)\cdot u)\ div\ ts(po(2,2),2),\ np(2)\}$

This case is more difficult, because although C_1 is strictly decreasing, C_2's values are not increasing. The only algorithm we could find was to try out all the $np(1)$ possible values of u and choose the one which minimizes the makespan. A similar situation occurs for the case $((1^u,tw,2^{np(1)-u}), (2^v,1^{np(2)-v}))$.

V. Minimum Weighted Sum of Completion Times

In this section we consider the objective of minimizing the weighted sum of completion times (given a weight $w(i)$ for each flow $i=1,2$):

$$ST = w(1)\cdot tfinish(1,np(1)) + w(2)\cdot tfinish(2,np(2)). \quad (7)$$

The techniques we used for determining the structure of minimum makespan schedules cannot be used here anymore. Despite this, we conjecture that the schedules which minimize the sum of completion times have the same structure as those minimizing the makespan and, thus, similar $O(np(i))$ optimization algorithms can be used. This is obvious for the simple cases $(k=1,\ P\geq 1)$ (where $ST=w(1)\cdot np(1)\cdot ts(po(1,1),1)+w(2)\cdot np(2)\cdot ts(po(2,1),2))$ and $(k \text{ does not exist},\ P=1)$ (where $ST=w(1)\cdot np(1)\cdot ts(1,1)+w(2)\cdot\ np(2)\cdot ts(1,2)\ +\ min\{w(2)\cdot np(1)\cdot ts(1,1),\ w(1)\cdot np(2)\cdot ts(1,2)\}$). We will consider two constrained versions of the problem, for which we provide dynamic programming algorithms.

A. Fixed Path for each Packet of Both Flows

We consider that the path on which each packet (i,j) will be sent is fixed. In this case, the minimum (weighted) sum of completion times is at least equal to:

$$ST_{low} = w(1)\cdot\sum_{j=1}^{np(1)} ts(path(1,j),1) + w(2)\cdot\sum_{j=1}^{np(2)} ts(path(2,j),2). \quad (8)$$

All we need to do is minimize the total weighted waiting time of the packets - caused by pairs of packets $(1,j(1))$ and $(2,j(2))$ scheduled on the same path and whose transit intervals might overlap. We will compute a table $T_{wait}(a,b)$=the minimum total weighted waiting time required for sending the first a packets of the first flow, the first b packets of the second flow and the packets $(1,a+1)$ and $(2,b+1)$ are scheduled to be sent at the same time moment. Initially, we have $T_{wait}(0,0)=0$ and $T_{wait}(a,b)=+\infty$ (for $a>0$ or $b>0$). We will use a forward type of dynamic programming. The pairs (a,b) $(0\leq a<np(1),\ 0\leq b<np(2))$ will be traversed in lexicographic order. If $T_{wait}(a,b)<+\infty$, then we will perform the following actions: we will advance forward in time, until all the packets of one of the two flows are sent or until a conflict occurs (packets $a'>a$ and $b'>b$ are scheduled on the same path and during overlapping time intervals). In the first situation, we will consider updating the minimum weighted sum of completion times by the value $ST_{low}+T_{wait}(a,b)$. In the second case, we will update the values $T_{wait}(a'-1,b')$ and $T_{wait}(a',b'-1)$.

MinimumWST-FixedPathsBothFlows():

```
ST=+∞ ; compute ST_low ; initialize T_wait(*,*)
for a=0 to np(1)-1 do
  for b=0 to np(2)-1 do
    if (T_wait(a,b)<+∞) then
      a'=a+1; b'=b+1
      tsa'=0; tsb'=0
      while ((a'≤np(1)) and (b'≤np(2))) do
        if (path(1,a')=path(2,b')) then break
        if (tsa'+ts(path(1,a'),1)<tsb'+ts(path(2,b'),2)) then
          tsa'= tsa'+ts(path(1,a'),1) ; a'=a'+1
        else if (tsa'+ts(path(1,a'),1)>tsb'+ts(path(2,b'),2)) then
          tsb'=tsb'+ts(path(2,b'),2) ; b'=b'+1
        else
          tsa'= tsa'+ts(path(1,a'),1) ; a'=a'+1
          tsb'=tsb'+ts(path(2,b'),2) ; b'=b'+1
      if ((a'>np(1)) or (b'>np(2))) then ST=min{ST,ST_low+T_wait(a,b)}
      else
        T_wait(a'-1,b')=min{ T_wait(a'-1,b'), T_wait(a,b)+w(1)·
                                    (tsb'+ts(path(2,b'),2)-tsa')}
        T_wait(a',b'-1)=min{ T_wait(a',b'-1), T_wait(a,b)+w(2)·
                                    (tsa'+ts(path(1,a'),1)-tsb')}
```

The time complexity is $O(np(1)\cdot np(2)\cdot(np(1)+np(2)))$.

B. Fixed Path for each Packet of Flow 1

In this case only the paths of the packets of the first flow are fixed. We need to minimize the sum of weighted waiting times plus the sum of weighted sending times of the packets of the second flow. We will use dynamic programming in a similar manner to the previous case and compute a table $T_{min}(a,b)$=the minimum total weighted time required for sending all the packets of the first flow, the first b packets of the second flow and the packets $(1,a+1)$ and $(2,b+1)$ are scheduled to be sent at the same time. We have $T_{min}(0,0)= w(1)\cdot(ts(path(1,1),1)+ts(path(1,2),1)+\ldots+ts(path(1,np(1)),1))$. For the other pairs (a,b), we initially have $T_{min}(a,b)=+\infty$. For each pair (a,b) with $T_{min}(a,b)<+\infty$, in lexicographic order, we will generate the

longest possible schedule (up to (a',b'), $a'\geq a$, $b'\geq b$), by scheduling the next type *2* packet on the fastest path which does not generate any conflicts. At each step, we also consider scheduling the type *2* packets on faster paths, thus obtaining conflicts – in this case, we try to update values like $T_{min}(a'',b'-1)$ and $T_{min}(a''-1,b')$, $a''\geq a'$:

MinimumWST-FixedPathsFlow1(P_{max}(≤P)):

$ST=+\infty$; $T_{min}(0,0)=w(1)\cdot(ts(path(1,1),1)+...+ts(path(1,np(1)),1))$
for $a=0$ **to** $np(1)-1$ **do**
 for $b=0$ **to** $np(2)-1$ **do**
  if $(T_{min}(a,b)<+\infty)$ **then**
   $a'=a+1$; $b'=b+1$
   $tsa'=0$; $tsb'=0$
   while $((a'\leq np(1))$ **and** $(b'\leq np(2)))$ **do**
    $bestp=uninitialized$
    for $p=1$ **to** P_{max} **do**
     $a''=a'$; $tsa''=tsa'$; $noconflict=true$
     while $((a''\leq np(1))$ **and** $(tsa''<tsb'+ts(po(2,p),2))$ **do**
      if $(path(1,a'')=p)$ **then**
       $T_{min}(a'',b'-1)=min\{T_{min}(a'',b'-1), T_{min}(a,b)+w(2)\cdot (tsa''+ts(path(1,a''),1))\}$
       $T_{min}(a''-1,b')=min\{T_{min}(a''-1,b'), T_{min}(a,b)+w(1)\cdot (tsb'+ts(po(2,p),2)-tsa'')+w(2)\cdot(tsb'+ts(po(2,p),2))\}$
       $noconflict=false;$ **break**
      else
       $tsa''=tsa''+ts(path(1,a''),1)$; $a''=a''+1$
     if $((bestp=uninitialized)$**and**$(noconflict=true))$ **then** $bestp=p$
    if $(bestp\neq unintialized)$ **then**
     $tsb'=tsb'+ts(po(2,bestp),2)$; $b'=b'+1$
     while $((a'\leq np(1))$ **and** $(tsa'+ts(path(1,a'),1)\leq tsb'))$ **do**
      $tsa'=tsa'+ts(path(1,a'),1)$; $a'=a'+1$
    else break
   if $(a'>np(1))$ **then** $T_{min}(np(1),b'-1)=min\{T_{min}(np(1),b'-1), T_{min}(a,b)+w(2)\cdot tsb'\}$
   if $(b'>np(2))$ **then** $T_{min}(a'-1,np(2))=min\{T_{min}(a'-1,np(2)), T_{min}(a,b)+w(2)\cdot tsb'\}$
for $a=0$ **to** $np(1)$ **do** $ST=min\{ST,T_{min}(a,np(2))\}$
for $b=0$ **to** $np(2)$ **do**
 $ST=min\{ST,T_{min}(np(1),b)+ w(2)\cdot(np(2)-b)\cdot ts(po(2,1),2)\}$

The time complexity of the algorithm is $O(np(1)\cdot np(2)\cdot (np(1)+np(2))\cdot P\cdot np(1))$. By computing the minimum of the runs of this algorithm (with $P_{max}=4$) over all the $4^{np(1)}$ possible paths for the packets of the first flow, we obtain an exponential solution for the (unconstrained) minimum weighted sum of completion times problem. We notice that, in this case, the proof of Theorem *2* holds and we can consider only the best *4* paths for the packets of the 1^{st} flow.

VI. Minimizing Makespan Without Packet Ordering Constraints

In this section we remove the packet ordering constraints for the packets of the same flow, i.e. we can send several packets of the same type in parallel, on distinct paths. We are interested in scheduling the $np(1)+np(2)$ packets on the P paths in such a way that the the makespan is minimized.

We will binary search the value C_{max} of the makespan and perform a feasibility test for each candidate value. The feasibility test first computes for each path k the maximum number of packets of type *1* that can be sent on it during the time interval $[0,C_{max}]$, $pf_1(k)=int(C_{max}/ts(k,1))$ (i.e. integer division). We will now present a pseudo-polynomial algorithm, similar in nature to the classical solution to the knapsack problem. Each path k will correspond to several items with weight q $(0\leq q\leq pf_1(k))$ and profit $pr(k,q)=int((C_{max}-q\cdot ts(k,1))/ts(k,2))$. We will compute a table $Pr_{max}(k, w)$=the maximum profit of a subset of items whose total weight is w, considering only the first k paths. The feasibility test is described below:

KnapsackFeasibilityTest(C_{max}, np(1), np(2)):

for $k=1$ **to** P **do** $pf_1(k)=$***int***$(C_{max}/ts(k,1))$
for $w=0$ **to** $np(1)$ **do** $Pr_{max}(0,w)=0$
for $k=1$ **to** P **do**
 for $w=0$ **to** $np(1)$ **do**
  $Pr_{max}(k,w)=Pr_{max}(k-1,w)$
  for $q=0$ **to** $min\{pf_1(k), w\}$ **do**
   $pr(k,q)=$***int***$((C_{max}-q\cdot ts(k,1))/ts(k,2))$
   if $(Pr_{max}(k-1, w-q)+pr(k,q)>Pr_{max}(k,w))$ **then**

$Pr_{max}(k,w)=Pr_{max}(k-1, w-q)+pr(k,q)$
if ($Pr_{max}(P,np(1))\geq np(2)$) **then return** *"passed"*
else return *"failed"*

The feasibility test computes the maximum number of flow *2* packets that can be scheduled on the *P* paths, after scheduling the *np(1)* packets of the first communication flow. Once the optimal value of the makespan was determined, we can run the feasibility test again and use the $Pr_{max}(*,w)$ values in order to determine the actual schedule (the number of packets of each flow which are scheduled on each path). From an implementation point of view, we notice that the feasibility test can use only $O(np(1))$ memory, instead of $O(P \cdot np(1))$ for the Pr_{max} table. This is because at any time we only require the last two rows of the Pr_{max} matrix ($Pr_{max}(k)$ and $Pr_{max}(k-1)$) – all the previous rows can be discarded; thus, we can use only two arrays. In order to compute the actual schedule, however, we may require all the $O(P \cdot np(1))$ entries of the matrix. In order to avoid $O(P \cdot np(1))$ memory storage, we propose a scheme which uses only $O((g+P/g) \cdot np(1))$ memory, where *g* is a parameter. When computing the $Pr_{max}(*,*)$ entries, we only store the rows of the matrix which are divisible by *g* (rows *0*, *g*, $2 \cdot g$, …) plus the last row *P*. Thus, $O(P/g)$ rows will be stored. These rows will be used as checkpoints. When determining the solution, we need to move back from row *P* down to row *0*. This procedure is rather standard; however, it assumes that the previous row is accessible in memory. Let's assume that we are currently at some row *r* which is stored in memory and we require the values on the row *r-1*, which is not stored in memory. We will find the largest row $q<r$ such that row *q* is stored in memory and generate all the rows *q+1*, *q+2*, …, *r-1* (using the same dynamic programming algorithm). All the generated rows (at most *g*) will be stored in memory. Now we will be able to move from row *r* all the way down to row *q*, which will be the next row whose previous row (*q-1*) is not stored in memory. We will discard all the previously generated rows (*q+1,…,r-1*), find the largest row $q'<q$ which is stored in memory and generate the rows *q'+1, q'+2, …, q'-1* (at most *g*). We repeat the procedure until we reach row *0*. At any moment, there will be $O(g+P/g)$ rows in memory, at the expense of doubling the running time of the algorithm (because every row will be generated twice overall). By choosing $g=sqrt(P)$ (*sqrt(P)*=the square root of *P*), we obtain $O(sqrt(P) \cdot np(1))$ memory used.

VII. Error-Correcting Information

In this section we consider an optimization problem regarding the carrying of error-correcting information in the context of a single communication flow. We are given a flow composed of *n* packets which are sent sequentially, in increasing order of their numbers (*1,2,…,n*), and a cost *c(i)* for each packet *i*, representing the amount of extra error-correcting information which needs to be added to the packet, if the packet is *selected* for this purpose. We consider that the flow is *secure* if there are at least *k* packets containing error-correcting information among any *m* consecutive packets ($2 \leq k \leq m \leq n$). We want to determine a subset of packets which are selected for carrying error-correcting information, such that the flow is secure and the total cost of the selected packets is minimum.

We will use dynamic programming and compute the values $C_{min}(i,d(1), \ldots, d(k-1))$ ($0<d(1)<\ldots<d(k-1)<m$), representing the minimum total cost of a subset of selected packets, such that the flow consisting only of the packets *1, …, i* is secure, packet *i* is selected and the previous *k-1* selected packets are located at some positions *p(1), … , p(k-1)*, such that $i-p(j) \leq d(j)$ ($1 \leq j \leq k-1$). We add *m+1* fictitious, additional packets: *m* at the beginning, which we number by *-m+1*, *-m+2*, …, *0,* and one packet at the end, numbered by *n+1*. Each of the additional packets has cost *0*. We have $C_{min}(1,*,\ldots,*)=c(1)$. For $i>1$, we first consider every sequence *(d(1), …, d(k-1))*: if $i-d(1) \leq 0$, then $C_{min}(i, d(1), \ldots, d(k-1))=c(i)$; otherwise, we set $C_{min}(i, d(1), \ldots, d(k-1))=c(i)+C_{min}(i-d(1), d(2)-d(1), \ldots, d(k-1)-d(1), m-d(1))$. After this step, we consider every sequence *(d(1), …, d(k-1))* in increasing lexicographic order and set $C_{min}(i, d(1), \ldots, d(k-1))=min\{C_{min}(i, d(1), \ldots, d(k-1)), C_{min}(i, d(1)-1, \ldots, d(k-1)), C_{min}(i, d(1), d(2)-1, \ldots, d(k-1)), \ldots, C_{min}(i, d(1), \ldots, d(k-1)-1)\}$ (we consider $C_{min}(i, d(1), \ldots, d(k-1))=+\infty$, if $d(1)=0$ or $d(j)=d(j+1)$ for some $1 \leq j \leq k-2$). The minimum total cost of a subset of selected packets is $C_{min}(n+1, m-k+1, \ldots, m-1)$.

The time complexity is $O(n \cdot m^{k-1})$. For $k=1$ the previous algorithm does not work, but we can use a simpler approach. We compute $C_{min}(i)$=the minimum total cost of a subset of selected packets, such that the flow restricted to the packets *1,…,i* is secure and packet *i* is selected. If $i \leq m$, $C_{min}(i)=c(i)$. Otherwise, $C_{min}(i)=c(i)+min\{C_{min}(j) | i-j \leq m\}$. We have several approaches here. We can test every value of *j*, obtaining an $O(n \cdot m)$ time complexity, or we can use a segment tree [7] over the sequence of packets. Each leaf of the segment tree corresponds to a packet. The value of each leaf is initially $+\infty$. After computing $C_{min}(i)$, we set the value of the corresponding leaf to $C_{min}(i)$ and update the aggregate values stored in the leaf's ancestors (by setting them to the minimum value among their left and right sons). Computing $C_{min}(i)$ ($i>m$) requires a range query over the interval *[i-m,i-1]*, which computes the minimum value of a leaf corresponding to a packet in this interval. It is well-known that we can perform updates and range queries in $O(log(n))$ time. An even better approach is to maintain a sorted double-ended queue (deque) with the $C_{min}(i)$ values contained in the interval *[i-m,i-1]*. When we move to the next vertex *i+1*, the interval slides one position to the right. With this approach [8], we can compute all the values in $O(n)$ time overall.

VIII. Related Work

The problems we discussed in this paper are related to the flexible job shop scheduling problem [1,2], where there are several jobs, each of which is composed of a number of operations and the operations of a single job must be executed sequentially on the *m* available machines. Algorithms for minimizing the makespan of file (packet) transfers were presented in

[3,4], but they considered very different situations (e.g. divisible file sizes, identical or uniform paths). The objective of minimizing the sum of completion times of jobs or file transfers is very important and was considered in [5,6].

IX. Conclusions and Future Work

In this paper we discussed the problem of scheduling the transfer of packets belonging to two communication flows on multiple disjoint packet-type aware paths, with the objective of minimizing the makespan. We identified the set of special structures an optimal schedule may have and presented a packet scheduling algorithm. We also considered the objective of minimizing the weighted sum of completion times and presented dynamic programming algorithms for two situations. The results presented in this paper are mostly of theoretical interest, but the patterns we observed could be extended and used in a practical setting. As future work, we intend to find a makespan minimization scheduling algorithm which can handle efficiently all the sub-cases involving waiting times. An $O(log(np(i)))$ time complexity would be highly desirable, but any sublinear algorithm would be an improvement.

References

[1] K. Jansen, M. Mastrolilli, and R. Solis-Oba, "Approximation Algorithms for Flexible Job Shop Problems," Proc. of Latin American Theoretical Informatics, LNCS vol. 1776, 2000, pp. 68-77.

[2] S. Reza Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: a review," International Journal of Production Research, vol. 43, 2005, pp. 2895-2929.

[3] M. I. Andreica and N. Ţăpuş, "High Multiplicity Scheduling of File Transfers with Divisible Sizes," Proc. of the IEEE International Symposium on Consumer Electronics, IEEE Press, 2008.

[4] E. Coffman, Jr., M. Garey, D. Johnson, and A. Lapaugh, "Scheduling file transfers," SIAM J. Comput., vol. 14 (3), 1985, pp. 744-780.

[5] J. Y.-T. Leung and H. Zhao, "Minimizing Sum of Completion Times and Makespan in Master-Slave Systems," IEEE Transaction on Computers, vol. 55 (8), 2006, pp. 985-999.

[6] X. Lu, R. A. Sitters, and L. Stougie, "A class of on-line scheduling algorithms to minimize total completion time," Operations Research Letters, vol. 31 (3), 2003, pp. 232-236.

[7] M. I. Andreica and N. Ţăpuş, "Optimal Offline TCP Sender Buffer Management Strategy," Proc. of the International Conf. on Comm. Theory, Reliability, and Quality of Service (CTRQ), 2008, pp. 41-46.

[8] P. Berman, et al., "Fast Optimal Genome Tiling with Applications to Microarray Design and Homology Search," J. Comput. Biol., vol. 11, 2004, pp. 766-785.