Journal of Computing and Information
Vol. 1, pp. 559–574, 1994

Routing and Scheduling File Transfers in Packet-Switched Networks

Weizhen Mao and Rahul Simha
Department of Computer Science
College of William and Mary
Williamsburg, VA 23185
wm, simha@cs.wm.edu

Abstract

Data traffic in networks has always been dominated by file transfers, an observation that has motivated previous work in scheduling file transfers. While these previous research contributions address file transfers, they do not address the special considerations required of packet-switched networks, in particular the issues of routing and of simultaneous transmissions of files. In this paper, we generalize previous formulations of the problem to include routing and to allow for simultaneous transmissions of several files. A few theoretical results are presented.

Keywords: Data transfers, File transfers, Scheduling, Routing, Large scale networking applications, NP-Completeness, Algorithms.

1 Introduction

We consider the joint problem of scheduling and routing the transfers of very large files. Our research is motivated by the increasing size and ambition of future networking applications, many of which will result in the movement of very large amounts of data across networks. For example, scientific applications include the data movement requirements of many grand challenge problems such as geophysical image data [1]. In education and entertainment, multimedia files are expected to range in the thousands of gigabytes [3, 12]. Another example is provided by the simple quotidian requirement of system backups across networks [8]; many remote databases must be archived at some central site. All these examples are characterized by large data transfers in which the sizes of individual data fragments are known. Furthermore, the resources needed for some cooperation among network nodes in order to compute an efficient schedule of transfers will be small compared to the actual time taken in moving the files about, thus making the effort of finding a good schedule appealing.

We formulate and analyze a file transfer scheduling problem that addresses many shortcomings of previous formulations. Our formulation is constructed based on considerations particular to what is expected to be the major domain of application, packet-switched networks. Specifically, we augment previous models with a necessary practical aspect, a routing component of the problem, and we consider link models more appropriate to the transfer of large files in packet-switched mode. We show that, true to the difficult nature of the problem, many parametrically simplified versions of the general problem are NP-complete. In addition, many standard approximation algorithms perform poorly in the worst case. Our results lead to the conclusion that the incorporation of the slightest routing problem, and hence some added graph-theoretic complexity, vastly increases the complexity of the general problem.

The problem of scheduling file transfers has surprisingly received limited attention in the literature. We are aware only of a few publications on the subject [5, 6, 7, 13]. All these seminal contributions study variations of the scheduling problem, but none tackle the singularly special considerations of file transfers in packet-switched networks. For example, they do not consider the fundamental problem of selecting routes and hence, avoid the added graph-theoretic complexity imposed by routing. Their assumptions do not allow links to be used simultaneously by several sessions, a key feature of packet-switched networks. Finally, the above work is more suited to special-purpose networks such as packet-radio networks [4, 13].

We organize this paper as follows. In Section 2, we present a formal description of our problem. A survey of past research is then made possible in Section 3. Following this, our complexity results are presented in Section 4, and a discussion of approximation algorithms is found in Section 5. Concluding remarks appear in Section 6. Detailed proofs of the theorems are included in the Appendix.

2 Problem formulation

Our problem of interest, the *General File Transfer Scheduling* problem ($GFTS$) has the following components: a file transfer graph, a network graph, and a specification of a link model.

The *file transfer graph* $G_F = (V_F, A_F)$ describes the file transfer requirements. Each arc $a = \langle u, v \rangle \in A_F$ represents a file to be transferred from node u to node v; the weight $l(a)$ is the length of the file to be transferred. Nodes in V_F are also called *sources* or *destinations* of the transfers.

The *network graph* $G_N = (V_N, E_N)$, $V_N \supseteq V_F$, describes the network requirements. Each edge $e = (u, v) \in E_N$ represents a full-duplex communication link between nodes u and v; the weight $s(e)$ is the speed of the link. Each node $v \in V_N$ has weight $m(v)$, denoting the amount of available memory at v. Since we are interested in problems with very large files, we must consider the case in which network nodes may not be able to hold the entire file, thus defeating some potential parallelism in transferring files. For each $a = \langle u, v \rangle \in A_F$, let $R_N(a)$ denote the set of routes in G_N that file a may take to accomplish the transfer. In this paper, we will assume that R_N is not restricted and must itself be computed off-line, meaning that once a route is selected for a file transfer it cannot be changed. Nodes in $V_N - V_F$ are called *intermediate nodes*.

The *link model* prescribes how contention among files for a link resource is resolved. In previous studies, the *serial model* is used: at any given time, a link in E_N is used by only one file. In our formulation of the problem we also consider the more practical *shared model*, in which a link resource is packet-switched and hence, any number of files may be using a link simultaneously, each rotating the use of the link on a packet-by-packet basis rather than on a file-by-file basis (as in the serial model). For example, two files, each of unit length sharing a unit speed link, take two time units each to get across the link. Two files, where one has unit length and the other is of length two, require three time units to complete (two time units while sharing the link, then one time unit to service the remaining half of the second file).

The distinction between the serial and shared models may be made more precise via the next example. Consider a two-node network with nodes 1 and 2 in which n files of lengths $l_1 \leq l_2 \leq \ldots \leq l_n$ are to be transferred from node 1 to node 2. Let s denote the speed of the only link between the two nodes. In the serial model, the time to complete transmission of file j depends on the order of transmission of the files. Thus, if files are sent in the order $1, 2, \ldots, n$, then the transmission of file j is completed at time $\frac{1}{s}\sum_{i=1}^{j} l_i$. In the shared model, all files are simultaneously transferred and hence, initially, each file is transferred at the shared speed of $\frac{s}{n}$, until the smallest file is completed. Following this, the remaining $n-1$ files share the link, each encountering a speed of $\frac{s}{n-1}$. More precisely, if we let C_i denote the completion time of file i in the shared model in the above example, then $C_i = C_{i-1} + \frac{n-i+1}{s} \cdot (l_i - l_{i-1})$. It is easy to check that $C_n = \frac{1}{s}\sum_{i=1}^{n} l_i$ in this case. We will refer to the case with the serial model as $GFTS1$ and to the case with the shared model as $GFTS2$.

Our problem is simply to arrange for the files to be transferred in the most efficient manner. This arrangement involves determining for each file the route it will take from source to destination and the times at which it begins transmission on various links. In the serial model, we will use the conventional *makespan* of the schedule (the time required to complete the transfers of all files) as the cost function, whereas for the shared model, we will use the *congestion factor* (the worst delay across the most congested link) as the cost function. A specific definition of the congestion factor will be given later. It should be pointed out that the reason we choose congestion factor as the cost function for $GFTS2$ is that in high-speed networks where the propagation delay is not very large the cost of congestion usually dominates the transfer time. Figure 1 shows an example of $GFTS1$, in which we assume that $l(a) = 1$ for $a \in A_F$, $s(e) = 1$ for $e \in E_N$, and $m(v) = \infty$ for $v \in V_N$. Note that link $(v_2, v_7) \in E_N$ is used by the transfers of files f_3, f_4 and f_6. Since the example is a serial model, this usage cannot be shared.

Since the cost function in the shared model is the congestion factor rather than

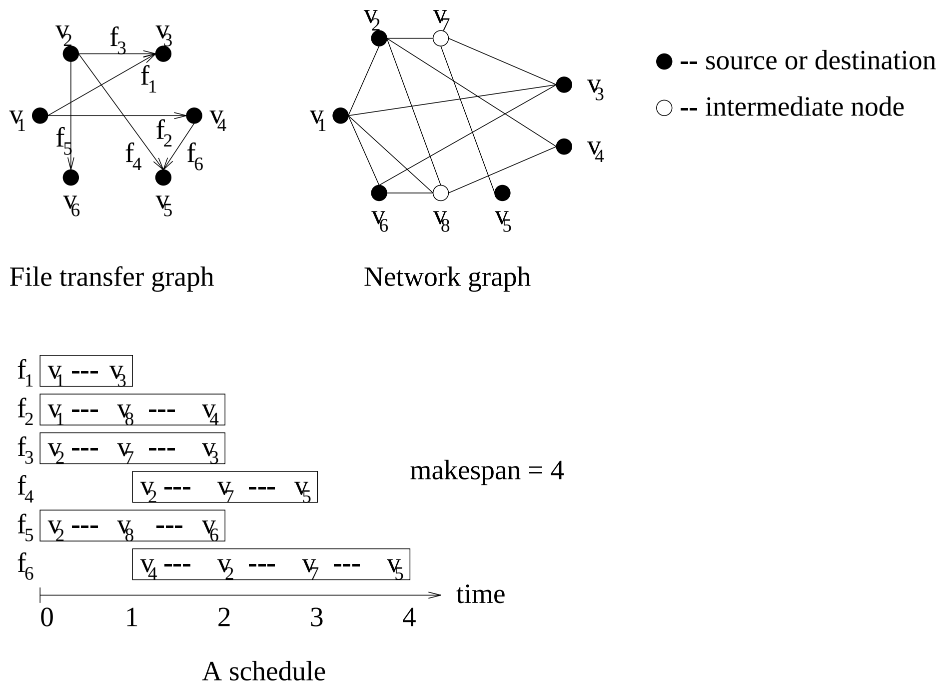

Figure 1: An example of $GFTS1$

the makespan, we can interpret $GFTS2$ without using the notion of time. Let $F = \{f_1, \ldots, f_n\}$ be the set of all files that need to be transferred, with lengths $l_1, l_2, \ldots, l_n$, respectively. Next define the assignment variable $\delta_{ie} = 1$ if and only if file f_i is routed over link e which has speed $s(e)$. Finally, construct a weighted graph $G_S = (V_S, E_S)$ where $V_S = V_N$, $E_S = E_N$ and the weight of each edge $e \in E_S$ is defined as

$$w(e) = \sum_{i=1}^{n} (\delta_{ie} \cdot \frac{l_i}{s(e)}).$$

Note that a proper schedule and selection of routes uniquely determines the assignment variables δ_{ie}. We then define the congestion factor to be the maximum edge weight in G_S. Therefore, minimizing the congestion factor in $GFTS2$ corresponds to minimizing the maximum edge weight in G_S. We will refer to the weighted graph G_S as the *time-relaxed graph* of the corresponding file transfer scheduling problem with shared model.

3 Previous work

The problem of scheduling file transfers was introduced by Coffman, Garey, Johnson and LaPaugh in their seminal paper on the same subject [7]. In this and subsequent work on file transfers [4, 5, 6, 13], the problem formulation is a restricted version of the one we have outlined in the following aspects:

- *Routing.* The additional complexity of routing is not considered.
- *Memory.* The memory problems associated with very large file sizes are not considered.
- *Port constraints.* The unsuitably restrictive constraint of limiting the number of concurrent sessions to a small fixed number is characteristic of earlier formulations.

Note that the problem of forwarding is considered in [13], although single routes are assumed ($\forall a \in A_F : |R_N(a)| = 1$).

In [7], the file transfer graph and network graph are the same, implying that sources and destinations have a direct link between them. Preemption is not allowed. Coffman *et al* show that their problem is NP-complete in general but also provide polynomial-time algorithms for some special cases. They also analyze very thoroughly the well-known *list scheduling heuristic* (which we also consider later), showing that in many cases, this simple technique often performs within a factor of 2 or 3 of the optimal solution. The paper mentions two open problems, both of which have been taken up in [6, 13].

We note that the original Coffman formulation restricted the network graph to the file transfer graph and also introduced the notion of a port constraint with the motivation that nodes represent machines with dialup modems. Home users would then have a port constraint of one, whereas large systems would have higher port capacities. The common graph would correspond to the telephone system. This model is also suited to point-to-point networks such as those constructed to operate in packet-radio format [2]. Such a framework is also considered by Choi in [4], in which the additional complexity of preemptive scheduling is allowed. Choi not only solves one of the open problems in [7] (determining whether odd-cycles can be solved in polynomial time) but also presents numerous useful results in the case that preemption is allowed. Again, some restrictions are NP-complete and some have polynomial-time solutions. Some of the contributions in [4] go beyond the file transfer problem to some general graph-theoretic results.

The other open problem mentioned in [7] is studied by Whitehead [13]: the case in which forwarding is permitted. A fixed route is prescribed via directed arcs for some source-destination pairs; the rest of the problem structure follows that in [7]. As one might expect, the added complexity results in the NP-completeness of some of the polynomially solvable versions of the original Coffman formulation. However, polynomial-time solutions are given for the special cases of bounded-degree trees and single-edge cycles.

As the reader may have already inferred, our formulation differs from the ones above both in structure as well as assumption. We explicitly allow for an arbitrary network connecting the network nodes and hence, include the further complexity of selecting good routes from sources to destinations. Note that a joint framework for routing and flow-control was studied in [9]; however, the formulation in [9] is essentially a routing problem modified to account for flow-control costs and does not have a scheduling component.

Next, in the context of packet-switched networks, we reject the port constraint defined in these earlier formulations; connections or sessions correspond to information streams and logically, there is no reason to limit the number of such connections. Note that we do not propose the use of preemption. We believe that preemption will very likely not be an option in future networks just as it is not an option now. This is because even lower-layer functions such as error control are expected to be performed end-to-end, leaving no possibility for preemptive control inside the network at a higher level.

4 Complexity results

In this section, we present complexity results for $GFTS1$ and $GFTS2$. If there exists an arc $a = \langle s_i, d_j \rangle \in A_F$ such that there is no route between s_i and d_j in G_N, then file a can never be transferred to its destination, and thus the cost of any schedule is ∞. We therefore assume that for any $a = \langle s_i, d_j \rangle \in A_F$, there is at least one route between s_i and d_j in G_N, i.e., $|R_N(a)| \geq 1$.

We say that a file transfer graph G_F is *bipartite* if V_F is partitioned into the set of sources and that of destinations and each arc connects a source to a destination. We say that a network graph is *bipartite* if there is no intermediate node, i.e., $V_N = V_F$, the set of network nodes is partitioned into the set of sources and that of destinations, and each edge (link) connects a source and a destination. For simplicity, in the following discussion we assume that G_F is always bipartite and that G_N is bipartite or of otherwise specified graph structure. We would like to point out that although the bipartite structure only represents a small group of packet-switched networks, the NP-completeness of the bipartite structure we shall establish below implies the NP-completeness of arbitrary graph structures. Due to the page limit of this paper, we will put the detailed proofs of the theorems presented in this section and the next section into the Appendix.

The first two theorems are about the complexity of $GFTS1$ and $GFTS2$ when the file lengths are arbitrary.

Theorem 1 *If G_F and G_N are both bipartite, files have arbitrary lengths and links have unit speeds, then $GFTS1$ is NP-complete.*

Theorem 2 *If G_F and G_N are both bipartite, files have arbitrary lengths and links have unit speeds, then $GFTS2$ is NP-complete.*

The next two theorems are about special cases of $GFTS1$ and $GFTS2$, which are solvable in polynomial time.

Theorem 3 *If G_F and G_N are both bipartite, then $GFTS1$ is polynomial when*

1. *G_F has no multi-arcs, or*
2. *G_N has no multi-edges, or*

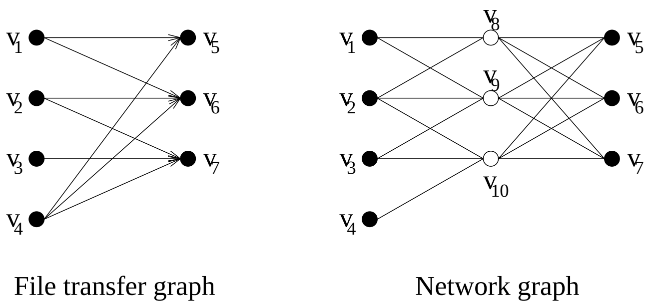

Figure 2: Extension of a bipartite graph

3. G_F *and* G_N *have multi-arcs and multi-edges, respectively, files have unit lengths and links have arbitrary speeds.*

Theorem 4 *If* G_F *and* G_N *are both bipartite, then* $GFTS2$ *is polynomial when*

1. G_F *has no multi-arcs, or*
2. G_N *has no multi-edges, or*
3. G_F *and* G_N *have multi-arcs and multi-edges, respectively, files have unit lengths and links have arbitrary speeds.*

Our next two results are a little more surprising. We consider the case in which a network graph differs only slightly from a bipartite graph by having an additional layer of intermediate nodes between sources and destinations, as is shown in Figure 2. We call this kind of graph *3-layered.*

Theorem 5 *If* G_F *is bipartite with only single-arcs,* G_N *is* 3*-layered with only single-edges, files have unit lengths and links have unit speeds, then* $GFTS1$ *is NP-complete.*

Theorem 6 *If* G_F *is bipartite with only single-arcs,* G_N *is* 3*-layered with only single-edges, files have unit lengths and links have unit speeds, then* $GFTS2$ *is NP-complete.*

Finally, we show that even for a simple network graph like a ring, $GFTS2$ remains to be NP-complete.

Theorem 7 *If* G_F *is bipartite with only single-arcs,* G_N *is a ring without intermediate nodes, files have arbitrary lengths and links have arbitrary speeds, then* $GFTS2$ *is NP-complete.*

5 Approximation algorithms

Since even the simplest versions of $GFTS1$ and $GFTS2$ are NP-complete, we next consider the design of fast approximation algorithms. In particular, we wish to examine standard scheduling heuristics like *list scheduling* [7, 10] and its variations. In list scheduling, a priority list of all files $(f_1, f_2, \ldots, f_n)$ is given and files are scheduled for transfer in the order the files appear in the priority list. Since the algorithm handles a file without knowing files following it in the list, list scheduling is in fact an *on-line* algorithm [11]. To evaluate the worst-case performance of the algorithm, we use the conventional *worst-case performance ratio* $\max_{\forall I}\{\frac{LS(I)}{OPT(I)}\}$, where $OPT(I)$ is the value of the cost function of the optimal file transfer schedule of instance I, and $LS(I)$ is the value of the cost function of the schedule of I obtained by the list scheduling algorithm. For simplicity, we assume that $m(v)$ is arbitrarily large. This assumption implies that node v can hold any number of files, and thus we do not have to consider the memory constraints while scheduling file transfers. We also assume that files have unit lengths and links have unit speeds. Consider the following method of creating a list schedule.

- List Scheduling 1 ($LS1$): For file $f_l = \langle s_i, d_j \rangle \in A_F$ $(l = 1, 2, \ldots, n)$, find any route between s_i and d_j in G_N, and then schedule f_l along this route.

In the serial model, a file stays at an intermediate node only when the link it chooses to use in the next time step is busy transferring a higher priority file, while in the share model, several files may share a link. We have the following theorem about the worst-case performance of $LS1$.

Theorem 8 *In both $GFTS1$ and $GFTS2$,* $\max_{\forall I}\{\frac{LS1(I)}{OPT(I)}\} \to \infty$ *as* $n \to \infty$.

We notice that $LS1$ chooses a route for a file arbitrarily. However, to minimize the cost function of the scheduling, we should avoid using any link too many times. With this goal in mind, we define the following modified list scheduling heuristic.

- List Scheduling 2 ($LS2$): For file $f_l = \langle s_i, d_j \rangle \in A_F$ $(l = 1, 2, \ldots, n)$, find a route between s_i and d_j in G_N with the fewest links that have been used before, and then schedule f_l along this route.

Theorem 9 *In both $GFTS1$ and $GFTS2$,* $\max_{\forall I}\{\frac{LS2(I)}{OPT(I)}\} \to \infty$ *as* $n \to \infty$.

We notice $LS2$ still cannot avoid heavy use of a certain link in some situations. What we want is a load balancing scheme that ensures even use of all links in the network. Let $num(e)$ be the number of times link $e \in E_N$ has been used so far. Define $\mu = \max_{\forall e}\{num(e)\}$. We then present another variation of the list scheduling algorithm.

- List scheduling 3 ($LS3$): For file $f_l = \langle s_i, d_j \rangle \in A_F$ $(l = 1, 2, \ldots, n)$, find a route between s_i and d_j in G_N with the smallest increase in μ, and then schedule f_l along this route.

Theorem 10 *In both $GFTS1$ and $GFTS2$,* $\max_{\forall I}\{\frac{LS3(I)}{OPT(I)}\} \to \infty$ *as* $n \to \infty$.

Thus, various versions of list scheduling ranging from the naive to the more complex all perform rather poorly in the worst case. However, the worst-case analysis may be overly pessimistic. A better performance measure is the average-case analysis. Although we do not have any analytic results, preliminary simulation experiments show that in practice both $LS2$ and $LS3$ outperform $LS1$ by a large margin while $LS3$ constructs file transfer schedules just a little better than $LS2$ does.

6 Conclusions

We have presented a general file transfer scheduling problem motivated by the anticipated large sizes of files in the applications of the future. We formulated an optimization problem that involves both scheduling as well as routing components. We have shown that various simplified versions of the general problem are NP-complete. Furthermore, the standard list scheduling algorithm performs poorly even when significantly augmented to account for link overlaps. Our main conclusion is therefore, the introduction of the slightest routing component vastly increases the complexity of the problem, even for approximation algorithms. Several research directions are evident: one, there may be better heuristics; two, there may be additional special cases which are solvable in polynomial time; and finally, there are considerations driven by networking applications, such as the need for a distributed approximation algorithm.

Acknowledgement

We wish to thank the three anonymous referees for their helpful comments and suggestions, and NSF for its financial support.

References

[1] G. Asrar and D. J. Dokken. *Earth Observing System Reference Handbook*. NASA Earth Science Support Office, Washington DC, 1993.

[2] D. Bertsekas and R. G. Gallager. *Data Networks*. Prentice Hall, 1992.

[3] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM SIGCOMM 93*, pp. 92–97.

[4] H.-A. Choi. Scheduling Data Transfers in Networks. *Ph.D. Thesis*, Northwestern University, 1985.

[5] H.-A. Choi and S. L. Hakimi. Scheduling Data Transfers in Networks with Preemptions. *Proc. 23rd Allerton Conference on Communications, Control and Computing*, 1985.

[6] H.-A. Choi and S. L. Hakimi. Scheduling File Transfers for Trees and Odd Cycles. *SIAM J. Computing*, **16**(1987), pp. 162–168.

[7] E. G. Coffman, M. R. Garey, D. S. Johnson and A. S. LaPaugh. Scheduling File Transfers. *SIAM J. Computing*, **14**(1985), pp. 744–780.

[8] G. D. Forney and B. Hajek. Research Priorities in Networking and Communications. *Report on NSF Workshop in Networking and Communications Research*, 1992.

[9] S. J. Golestaani. A Unified Theory of Flow Control and Routing on Data Communication Networks. *PhD Thesis*, Massachusetts Institute of Technology, 1980.

[10] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.*, **17**(1969), pp. 416–429.

[11] R. M. Karp. On-line Algorithms Versus Off-line Algorithms: How Much is it Worth to Know the Future? *Technical Report TR-92-044*, International Computer Science Institute, Berkeley, CA.

[12] Proceedings of the First International Workshop on Network and Operating System Support for Digital Audio and Video. TR-90-062, International Computer Science Institute, Berkeley, CA.

[13] J. Whitehead. The Complexity of File Transfer Scheduling with Forwarding. *SIAM J. Computing*, **19**(1990), pp. 222–245.

Routing and Scheduling File Transfers in Packet-Switched Networks — Appendix

Theorem 1 *If G_F and G_N are both bipartite, files have arbitrary lengths and links have unit speeds, then $GFTS1$ is NP-complete.*

Proof This can be proved by a simple reduction from the NP-complete *Multiprocessor Scheduling* (MS) problem.

In MS, we are given $m \geq 2$ identical processors, and n independent jobs of lengths $p_1, p_2, \ldots, p_n$, respectively. Let us define an instance for $GFTS1$ as follows: $G_F = (\{s,d\}, \{a_1, a_2, \ldots, a_n\})$, where $a_i = \langle s, d \rangle$ and $l(a_i) = p_i$, for $i = 1, 2, \ldots, n$. $G_N = (\{s,d\}, \{e_1, e_2, \ldots, e_m\})$, where $e_i = (s,d)$ and $s(e_i) = 1$, for $i = 1, 2, \ldots, m$. See Figure 3. It is easy to prove that there is a nonpreemptive multiprocessor schedule with makespan $\leq B$ if and only if there is a file transfer schedule for the instance defined above with makespan $\leq B$.

Since the reduction can be performed in polynomial time, and the problem is clearly in NP, the result follows. $\square$

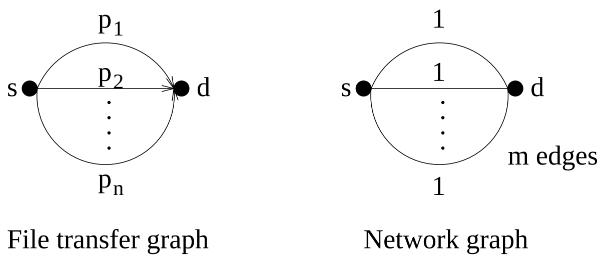

Figure 3: An instance for $GFTS1$

Theorem 2 *If G_F and G_N are both bipartite, files have arbitrary lengths and links have unit speeds, then $GFTS2$ is NP-complete.*

Proof Although the cost function for $GFTS2$ is the maximum edge weight in the corresponding time-relaxed graph G_S, the NP-completeness proof in Theorem 1 applies identically to $GFTS2$. $\square$

Theorem 3 *If G_F and G_N are both bipartite, then $GFTS1$ is polynomial when*

1. *G_F has no multi-arcs, or*
2. *G_N has no multi-edges, or*
3. *G_F and G_N have multi-arcs and multi-edges, respectively, files have unit lengths and links have arbitrary speeds.*

Proof First, consider the case that G_F has no multi-arcs. Since there is a link (route) in G_N for each file in G_F, all files can be transferred in parallel. If there is more than one link for a file, choose the fastest one. Therefore, the minimum makespan is

$$\max_{\forall a=\langle v_1,v_2\rangle \in A_F} \left\{ \frac{l(a)}{\max_{\forall e=(v_1,v_2)\in E_N}\{s(e)\}} \right\}.$$

It is not hard to see that it takes $O(|A_F|+|E_N|)$ time to compute the minimum makespan and construct the optimal schedule.

Next, consider the case that G_N has no multi-edges. Since G_F may have multi-arcs, files having the same sources and the same destinations are transferred sequentially, and files having different sources or different destinations are transferred in parallel. Define $G'_F = (V_F, A'_F)$ to be G_F with each multi-arc replaced by a single-arc of weight being the sum of those on the multi-arc. Therefore, the minimum makespan is

$$\max_{\forall a=\langle v_1,v_2\rangle \in A'_F} \left\{ \frac{l(a)}{s(e) \text{ for } e=(v_1,v_2)\in E_N} \right\}.$$

It is not hard to see that it takes $O(|A_F|)$ time to compute the minimum makespan and construct the optimal schedule.

Finally, consider the case that G_F and G_N have multi-arcs and multi-edges, respectively, files have unit lengths and links have arbitrary speeds. Files having the same sources and the same destinations make full use of the multiple links (routes) available in G_N. Therefore, the minimum makespan is

$$\max_{\forall \langle v_1, v_2 \rangle \in A_F} \{M^*(v_1, v_2)\},$$

where $M^*(v_1, v_2)$ is the minimum makespan to transfer n (the number of arcs from v_1 to v_2 in G_F) files of unit length using m (the number of edges between v_1, v_2 in G_N) links of arbitrary speeds. This is equivalent to the classical problem of scheduling n unit-length jobs on m uniform machines, which according to [2, 3] can be solved in $O(n^2)$ time. It is not hard to see that it takes $O(|A_F|^2)$ time to compute the minimum makespan and construct the optimal schedule. □

THEOREM 4 *If G_F and G_N are both bipartite, then $GFTS2$ is polynomial when*

1. *G_F has no multi-arcs, or*
2. *G_N has no multi-edges, or*
3. *G_F and G_N have multi-arcs and multi-edges, respectively, files have unit lengths and links have arbitrary speeds.*

Proof Although the cost function for $GFTS2$ is the maximum edge weight in the corresponding time-relaxed graph G_S, the algorithms described in the proof of Theorem 3 all work well for the corresponding cases of $GFTS2$. □

THEOREM 5 *If G_F is bipartite with only single-arcs, G_N is 3-layered with only single-edges, files have unit lengths and links have unit speeds, then $GFTS1$ is NP-complete.*

Proof Consider the NP-complete *Timetable Design* (TD) problem defined in [1]. Given a set of three work hours $H = \{h_1, h_2, h_3\}$, a set of craftsmen $C = \{c_1, c_2, \ldots, c_m\}$, and a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$. Let $A(c) \subseteq H$ be the available hours for $c \in C$, $A(t) = H$ be the available hours for $t \in T$, and $R(c, t) = 0, 1$ be the required work hour for $c \in C$ on $t \in T$. The problem asks to define a function $f : C \times H \times T \rightarrow \{0, 1\}$ such that (1) $f(c, h, t) = 1$ only if $h \in A(c)$; (2) $\forall c, h$, there is at most one t for which $f(c, h, t) = 1$; (3) $\forall h, t$, there is at most one c for which $f(c, h, t) = 1$; and (4) $\forall c, t$, there are exactly $R(c, t)$ values of h for which $f(c, h, t) = 1$.

Let us define an instance for $GFTS1$ as follows: $G_F = (C \cup T, \{(c, t) | R(c, t) = 1\})$, $G_N = ((C \cup T) \cup \{h_1, h_2, h_3\}, \{(c, h) | h \in A(c)\} \cup \{(h, t) | h \in A(t)\})$, and the upper bound B for makespan is 2. As we understand, $l(a) = 1$ for $a \in A_F$, and $s(e) = 1$ for $e \in E_N$. For example, if $H = \{h_1, h_2, h_3\}$, $C = \{c_1, c_2, c_3, c_4\}$, $T = \{t_1, t_2, t_3\}$, $A(c_1) = \{h_1, h_2\}$, $A(c_2) = \{h_1, h_2, h_3\}$, $A(c_3) = \{h_2, h_3\}$, $A(c_4) = \{h_3\}$, and $R(c, t)$ is defined to be 1 except that $R(c_1, t_2)$, $R(c_2, t_1)$, $R(c_3, t_1)$ and $R(c_3, t_2)$ are 0, then the corresponding instance for $GFTS1$ is shown in Figure 4.

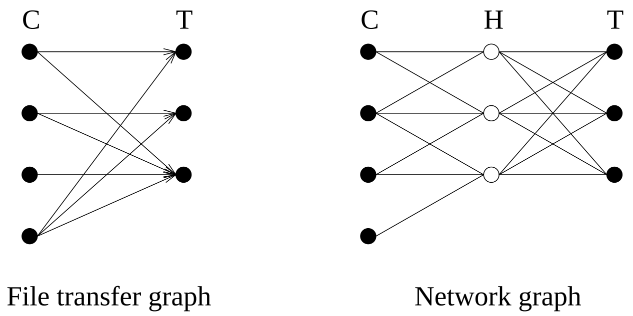

Figure 4: An instance for $GFTS1$ with one layer of intermediate nodes

We claim that TD has a timetable if and only if there is a file transfer schedule for the instance defined above with makespan 2. Assume that TD has a timetable, i.e., there is $f : C \times H \times T \to \{0, 1\}$. Consider all triples (c, h, t) with $f(c, h, t) = 1$, each of which corresponds to a route in G_N. Because of requirements (2) and (3) in the definition of TD, these routes are all edge-disjoint. Furthermore, because of requirement (4), for each file $(c, t) \in A_F$, there is exactly one route. This indicates that all files in G_F can be transferred in parallel. Therefore the makespan is 2. Assume that the instance for $GFTS1$ has a schedule of makespan 2, i.e., there are edge-disjoint routes in G_N, one for each file in G_F. We define f such that $f(c, h, t) = 1$ if and only if route (c, h, t) in G_N is used to transfer file (c, t). It is obvious that requirement (1), (2), (3), and (4) are all satisfied. Therefore, TD has a timetable. We conclude that $GFTS1$ is NP-complete. $\square$

Theorem 6 *If G_F is bipartite with only single-arcs, G_N is 3-layered with only single-edges, files have unit lengths and links have unit speeds, then $GFTS2$ is NP-complete.*

Proof If we change the bound B to 1, the proof of Theorem 5 applies identically to $GFTS2$. $\square$

Theorem 7 *If G_F is bipartite with only single-arcs, G_N is a ring without intermediate nodes, files have arbitrary lengths and links have arbitrary speeds, then $GFTS2$ is NP-complete.*

Proof We show NP-completeness by a reduction from the following NP-complete problem: given a set of positive numbers $A = \{a_1, \ldots, a_n\}$ and a positive number a, is there a partition A_1, A_2 of A such that $\max\{\sum_{a_i \in A_1} a_i, \sum_{a_i \in A_2} a_i\} \leq a$?

Given an instance of the above problem, construct an instance of $GFTS2$ as in Figure 5. The bipartite file transfer graph G_F simply consists of sending a file of length a_i from node i to node $n + i$, i.e., arc $\langle i, n + i \rangle$ has weight a_i for $i = 1, 2, \ldots, n$. The network graph is a ring of $2n$ nodes, where links $(1, n + 1)$ and $(n, 2n)$ have unit speeds while other links in the ring have arbitrarily high speeds.

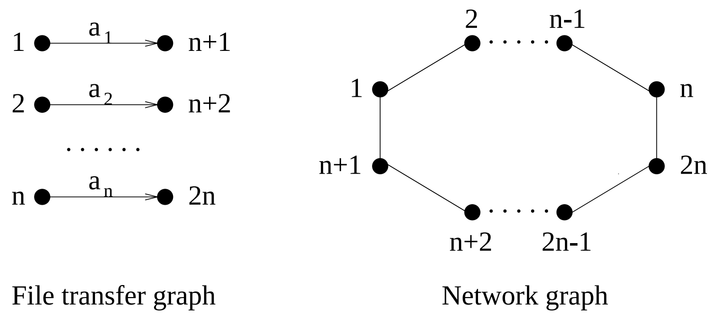

Figure 5: An instance of $GFTS2$ when G_N is a ring

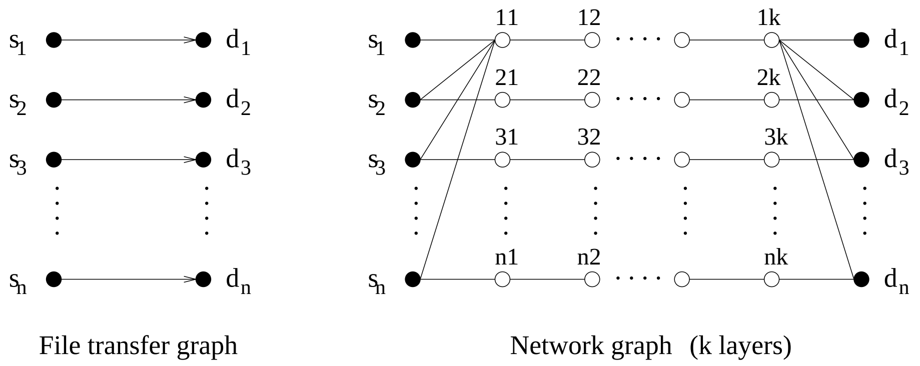

Figure 6: An instance for $LS1$

Consider the time-relaxed graph G_S for the instance. We observe that in G_S either edge $(1, n+1)$ or edge $(n, 2n)$ will have maximum weight among all edges in G_S because all the delay will occur on these two slow links. We also observe that every file must use exactly one of these two edges. Since the cost function in $GFTS2$ is the maximum edge weight in G_S, it will be less than or equal to a if and only if $\max\{\sum_{a_i \in A_1} a_i, \sum_{a_i \in A_2} a_i\} \leq a$, which completes the proof. $\square$

THEOREM 8 *In both $GFTS1$ and $GFTS2$, $\max_{\forall I}\{\frac{LS1(I)}{OPT(I)}\} \to \infty$ as $n \to \infty$.*

Proof Let us consider the instance I in Figure 6, where the network graph has k layers of intermediate nodes.

Clearly, in the optimal schedule, all n files are transferred in parallel, i.e., file $f_i = \langle s_i, d_i \rangle$ uses route $s_i \to i1 \to i2 \to \cdots \to ik \to d_i$, for $i = 1, 2, \ldots, n$. In the serial model, since it takes $k+1$ steps to travel from a source to a destination in the network graph, $OPT(I) = k+1$, while in the shared model, since each link is used by at most one file, $OPT(I) = 1$. Assume that the priority list is $(\langle s_1, d_1 \rangle, \langle s_2, d_2 \rangle, \ldots, \langle s_n, d_n \rangle)$. In the worst case for $LS1$, which may be achieved by certain implementation of $LS1$, all n files use the topmost route of intermediate nodes $11 \to 12 \to \cdots \to 1k$. Therefore, in the

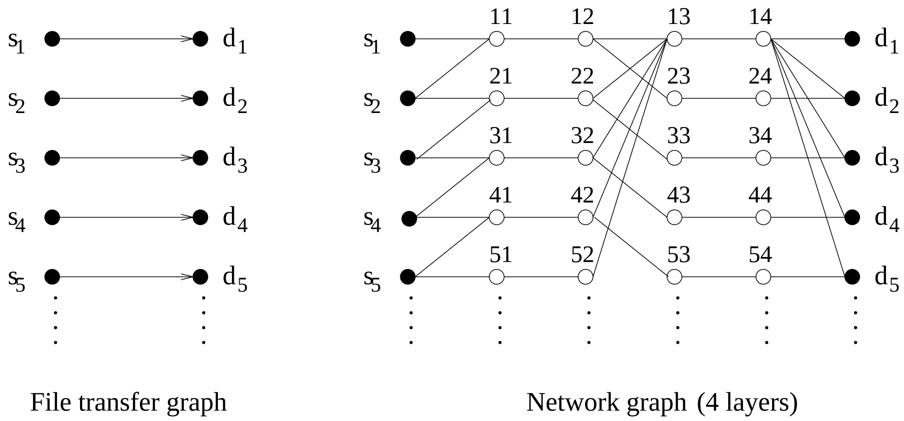

Figure 7: An instance for $LS2$

serial model, $LS1(I) = (k+1)+(n-1) = k+n$, while in the shared model, $LS1(I) = n$. So $\max_{\forall I}\{\frac{LS1(I)}{OPT(I)}\} \to \infty$ as $n \to \infty$ in both models. □

Theorem 9 *In both $GFTS1$ and $GFTS2$, $\max_{\forall I}\{\frac{LS2(I)}{OPT(I)}\} \to \infty$ as $n \to \infty$.*

Proof Let us consider the instance I in Figure 7, where the network graph has 4 layers of intermediate nodes.

Clearly, in the optimal schedule, file $f_1 = \langle s_1, d_1\rangle$ uses route $s_1 \to 11 \to 12 \to 13 \to 14 \to d_1$, and file $f_i = \langle s_i, d_i\rangle$ uses route $s_i \to (i-1)1 \to (i-1)2 \to i3 \to i4 \to d_i$, for $i = 2, 3, \ldots, n$. Only link $(11, 12)$ in G_N is used twice. Therefore, $OPT(I) = 5+1 = 6$ in the serial model, and $OPT(I) = 2$ in the shared model. Assume that the priority list is $(\langle s_1, d_1\rangle, \langle s_2, d_2\rangle, \ldots, \langle s_n, d_n\rangle)$. In the worst case for $LS2$, file $f_1 = \langle s_1, d_1\rangle$ uses route $s_1 \to 11 \to 12 \to 13 \to 14 \to d_1$, and file $f_i = \langle s_i, d_i\rangle$ uses route $s_i \to i1 \to i2 \to 13 \to 14 \to d_i$, for $i = 2, 3, \ldots, n$. Therefore, in the serial model, $LS2(I) = 5+(n-1) = n+4$, while in the shared model, $LS2(I) = n$. So $\max_{\forall I}\{\frac{LS2(I)}{OPT(I)}\} \to \infty$ as $n \to \infty$ in both models. □

Theorem 10 *In both $GFTS1$ and $GFTS2$, $\max_{\forall I}\{\frac{LS3(I)}{OPT(I)}\} \to \infty$ as $n \to \infty$.*

Proof Let us consider the instance I in Figure 8, where the network graph has 1 layer of intermediate nodes and $n = 1+2+\cdots+h$.

Clearly, in the optimal schedule, all n files are transferred in parallel, i.e., file $\langle s_i, d_{ij}\rangle$ uses route $s_i \to i(j+1) \to d_{ij}$ if $i \neq j$, and it uses route $s_i \to i1 \to d_{ij}$ if $i = j$. Therefore, $OPT(I) = 2$ in the serial model, and $OPT(I) = 1$ in the shared model. Assume that the priority list is $(\langle s_1, d_{11}\rangle, \langle s_2, d_{21}\rangle, \langle s_2, d_{22}\rangle, \ldots, \langle s_h, d_{hh}\rangle)$. In the worst case of $LS3$, file $\langle s_i, d_{ij}\rangle$ uses route $s_i \to i1 \to d_{ij}$. To better understand this, note that when all files whose sources have indices less than or equal to some constant k are completed, $\mu = k$. To transfer the files with sources of index $k+1$, we can choose any route as long as the

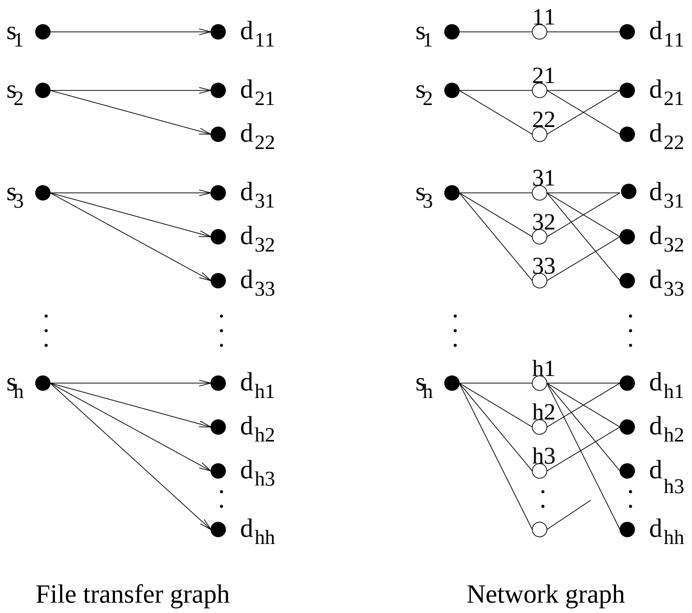

Figure 8: An instance for $LS3$

increase of μ is minimal. Therefore we can use route $s_{k+1} \rightarrow (k+1)1 \rightarrow d_{(k+1)j}$ for file $\langle s_{k+1}, d_{(k+1)j} \rangle$ with $j \leq k$, keeping μ unchanged. However, when it comes to transfer $\langle s_{k+1}, d_{(k+1)(k+1)} \rangle$, we have no choice but to use route $s_{k+1} \rightarrow (k+1)1 \rightarrow d_{(k+1)(k+1)}$, thus increasing μ by 1 to $k+1$. Therefore, $LS3(I) = 2 + (h-1) = h+1$ in the serial model, and $LS3(I) = h$ in the shared model. So $\max_{\forall I}\{\frac{LS3(I)}{OPT(I)}\} \rightarrow \infty$ as $n \rightarrow \infty$ in both models. $\square$

References

[1] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[2] E. L. Lawler. Optimal Sequencing of a Single Machine Subject to Precedence Constraints. *Management Sci.*, **19**(1973), pp. 544–546.

[3] E. L. Lawler. Unpublished.