

# Optimizing deadline-driven bulk data transfers in overlay networks

Andrei Agapi\*, Sebastien Soudan†, Marcelo Pasin‡, Pascale Vicat-Blanc Primet† and Thilo Kielmann\*

\*Vrije Universiteit, Dept. of Computer Science, Amsterdam, The Netherlands

†Ecole Normale Supérieure de Lyon, CNRS-INRIA-ENS LYON-UCBL 5668, France

‡Universidade de Lisboa, Dept. Informática, Lisbon, Portugal

Contact email: aagapi@few.vu.nl

**Abstract**—Deadline-driven bulk data transfers frequently occur in overlay networks running data-intensive, distributed workflow applications, such as grid and cloud environments. What distinguishes such transfers from other Internet traffic is that overlay nodes should cooperate towards the common goal of delivering all inter-dependent data timely, rather than follow individual, selfish goals. For such scenarios, we propose scheduling transfers in overlays in a globally optimal manner with respect to minimizing overall network congestion. Our optimization jointly addresses routing of transfers within the overlay and the time-domain scheduling of transfer bandwidths. We formally define and address the associated problem, the Bulk Data Routing and Transfer (BDRT) and present a linear programming-based solution to it, optimal in both routing and time domains. We additionally explore alternative approaches based on heuristic routing strategies, both oblivious and time-domain optimized. We evaluate these solutions via both PlanetLab trace-driven simulations and Internet transfer experiments, on the Intrigger wide-area grid and PlanetLab. Evaluation shows that our approach finds optimal solutions, based on estimations of job arrival times, deadlines and transfer volumes.

**Index Terms**—congestion minimization, deadline-driven transfers, routing overlays, Internet

## I. INTRODUCTION

In the present work, we address the problem of scheduling bulk data transfers in routing overlays in a globally optimal manner. In the context of a best-effort Internet, unable to provide QoS guarantees for applications, systems such as opportunistic routing overlays [1][11][7] have emerged over the past decade to provide a flexible and interesting way to get extra QoS out of public networks by only manipulating the application level, with no cooperation from the underlying network. Such overlays typically attempt to improve QoS by routing around congestion and network bottlenecks. Besides such systems, many other overlays and p2p systems such as Bittorrent, whether in a structured or heuristic manner, as a design goal or as a side effect of their design, use various forms of application-level relaying or multicasting in an attempt to improve network performance for the peers using them. While this approach might be profitable for individual, selfish peers,

a question arises on the efficiency of these approaches from the perspective of the network as a whole. Taking such a perspective might be interesting from the point of view of emerging applications, such as data intensive p2p computing, cloud computing, distributed stream processing and, in general, for any distributed application in which distant peers have common, rather than selfish goals. Examples include commercial distributed organizations that need to transfer and process data produced in real-time at remote locations or soft real-time, SETI@home-like processing of large scale data. In such setups, data sources, as well as processing elements may be widely distributed and workflows may imply multiple inter-related high bandwidth streams sharing the same network links. These streams can be inter-dependent because of application logic or may need to be synchronized. For these scenarios, cooperatively transferring the data in a globally optimal manner, such that all of the deadlines are met, is a much more appropriate approach than acting as a collection of selfish peers. Even in the case of applications where peers have selfish goals (such as file downloads), a global optimization might, in some instances, yield better results than current approaches, whereby each peer generally uses the system in a greedy manner. We propose the global optimization of bulk transfers in an overlay in two dimensions: the time dimension and the routing dimension. The time dimension refers to scheduling each requested transfer in the time domain such that the deadlines for all requests are met, and additionally a global objective (in the current work, network congestion) is optimized. Basically, in each time interval the bandwidth that each transfer can use is rate-limited such that the global objective is cooperatively achieved. The routing dimension refers to optimally routing requests within the overlay with the same objective, congestion minimization. In the context of public networks characterized by cross-traffic and variable available bandwidth, minimizing network congestion (i.e. the maximum link utilization on any link in the network) basically means we minimize the ratio of available bandwidth our application *requires*

from network links to achieve its goals. A desirable effect of this approach is that we actually maximize the probability that the required value is really available on those links (intuitively, on any link, it is more likely that at least 200 Kbps are available than 1 Mbps), therefore, the likelihood that *all* transfer deadlines are met is improved. The problem we address, that we call Bulk Data Routing and Transfer (BDRT), is to jointly optimize routing within the overlay, as well as the time-domain transfer schedules for the transfer requests, such that all deadlines are met and the overall network congestion is minimized. For now, we address network congestion minimization at overlay link (rather than native link) layer. The paper is structured as follows. Section II defines and formulates the BDRT problem. We describe our solution in Section III. We propose an optimal, linear programming-based approach (III-A) and compare it to approaches based on heuristic routing (III-B). In Subsections III-C and III-D, we describe the routing and traffic shaping overlay infrastructure we developed. Section IV presents our evaluation results. Section V addresses related work and Section VI concludes.

## II. BACKGROUND AND PROBLEM DEFINITION

The paradigm we work in is the one previously defined and used in work such as [4] or [2]. Bulk transfers are characterized, besides the data volume  $\nu_r$  to be transmitted, by an arrival time  $\eta_r$  (at which data is available at the source) and a deadline  $\phi_r$  (by which all data has to be transferred to the destination). In the offline scheduling case, the simplest case, we assume that all the information on future transfer requests can somehow be estimated ahead of time, which can be the case for many applications. We define the interval  $\omega_r = [\nu_r, \phi_r]$  as the active window of a request. Informally, we define the Bulk Data Routing and Transfer (BDRT) problem as: given a network graph and a finite set of transfer requests described as tuples (transfer volume, active window), find the network paths to route each transfer on, as well as the bandwidth allocation profile  $\lambda_r$  for each of these requests and each of these paths such that all deadlines are met and the overall network congestion is minimized. We define a bandwidth allocation profile as a function  $\lambda_r : \omega_r \rightarrow R+$ , specifying the rate at which request  $r$  is entitled to transfer at time  $t$ . [2] has shown that, by dividing the timeline in  $2|T| - 1$  intervals generated by all transfer arrival times and deadlines, where  $|T|$  is the number of requests (similarly to [9]), the problem defined has an optimal solution with the bandwidth allocation profile for each task in the form of a step function with  $O(|R|)$  intervals. This means that, within each interval, the rate at which the transfer bandwidth is limited, for any task, is a constant. In the current work, we find that BDRT is in P and effectively

solvable by linear programming (e.g. by the simplex or interior point methods). The solution to BDRT can be given as a set of so-called multipath profiles, one per each request, i.e. a function  $\mu_r : L \times \omega_r \rightarrow R+$ , where  $L$  is the set of links in the network. This function basically gives us the set of links used by request  $r$  at each time interval as well as the rate it is entitled to transfer at time  $t$  on each of those links. The set of links used in interval  $i$  are those  $l$  for which  $\mu_r(l, i) \neq 0$ . Next, we formally define BDRT.

**Definition 1.** A single link bandwidth profile is a step function  $\lambda(\omega_i) = \alpha_i$ , where  $\omega \in \Omega$  is a set of time intervals, and  $\alpha \in A$  is a set of constant values defining  $\lambda$  on each interval  $\omega_i$ .

**Definition 2.** An overlay network is represented by an oriented graph  $G(N, L)$ , consisting of a node set  $N$  and a link set  $L$ . A node  $n \in N$  is a vertex in the graph  $G$  which, for a given request, can be a sender node, a receiver node or a relay node, forwarding traffic. A link is a directed edge on  $G$ , defined as  $l = \langle s_l, d_l, cap_l \rangle \in L$ , characterized by a source node  $s_l$ , a destination node  $d_l$ , and a capacity  $cap_l$ .

**Definition 3.** A request  $r = \langle s_r, d_r, \nu_r, \eta_r, \phi_r \rangle \in N \times N \times R \times subsets(I)$ , is defined by a source node  $s_r$ , a destination node  $d_r$ , a volume to transfer  $\nu_r$ , an arrival time  $\eta_r$ , and a deadline  $\phi_r$ . The active window of  $r$  is defined as the interval  $\omega_r = [\nu_r, \phi_r]$ . We denote by  $T$  the set of all requests available as input to the scheduling problem.

**Definition 4.** The interval set of a request  $r$  is  $\Omega_r = \{\omega_r\}$ . Informally,  $\Omega_r$  is the set of those time intervals as defined by the division in  $2|R| - 1$  intervals given by the arrival times and deadlines of the  $|R|$  jobs, that fall within the active window of  $r$ . The global interval set of transfer request  $T$  is defined as  $\Omega_T = \bigcup_{r \in T} \Omega_r$ .

**Definition 5.** For a link  $l \in L$  and a time interval  $i \in I$ , we define  $k_{l,i} \in R+$  as the utilization of that link in that specific time interval, i.e. the sum of bandwidths that all requests  $r \in T$  have scheduled on that link in that interval divided by the link's capacity. We define  $h_l = \max_{i \in I}(k_{l,i})$ , i.e. the maximum utilization of link  $l$  on any time interval and  $g = \max_{l \in L}(h_l)$ , i.e. the maximum link utilization on any link in the network.  $g$  is often referred to as the network congestion factor and will be our minimization objective variable.

For  $k_{l,i} > 1$ , we say that link  $l$  is overbooked. BDRT has a feasible solution if  $g \leq 1$ , i.e. all deadlines will be met. For  $g > 1$  we say that the network is overbooked. In our implementation and evaluation, we allow link utilizations to go above 1, since we consider  $g$  a relevant quality metric for the scheduling algorithms, i.e. a smaller  $g$  is always better, even if

it is greater than 1, since it will supposedly cause deadlines to be missed by less time than a higher  $g$ . Let us note that in the context of public networks with cross-traffic, capacity  $cap_l$  would represent an expected value of the available bandwidth on that link (further details on deriving this expected value in subsection III-D). Regarding the complexity of BDRT, referring to [5], where an extensive analysis of various flow over time problems is presented, we find that we can regard BDRT as an instance of the "min-cost multicommodity flow over time (MCFT) with uniform path-lengths" problem, which Theorem 1 of this paper proves to be polynomial. The reduction holds for BDRT if we ignore the "transit times" of the network links, which well approximates reality if we consider the bulk data transfer problem to be dominated by link bandwidths and we ignore latencies. By this we do not mean that latency does not influence throughput (e.g. by longer RTT of TCP ACK packets causing longer retransmission times of lost packets), but that the "transit time on pipes", as defined in [5] can be considered negligible in the context of long-lived Internet data transfers. While we consider transit times negligible (thus uniform) in BDRT, the MCF problem does not consider requests with variable start times and deadlines. However, a reduction can be straightforwardly achieved by adding two extra graph vertices per each request, before the source and after the destination, with infinite capacity and transit times equal to the request arrival time and the difference deadline - arrival time, respectively. This keeps path lengths uniform w.r.t. transit times and the problem equivalent. BDRT can thus be reduced to a polynomial problem.

### III. SOLUTION DESCRIPTION

#### A. The linear programming approach

Furthermore, we find that the problem can be written in a (pure) LP form, inspired by Wang's work in [13]. Done in the context of Internet core traffic engineering, [13] solves the routing problem for the case in which transfer requests arrive simultaneously and there are no deadlines or data volumes characterizing transfers, but rather each request is treated as a continuous bandwidth reservation, typical for quasi-infinite Internet core flows. Analogously to [13], we propose a linear optimization method for BDRT, in which requests are additionally characterized by variable data volumes, arrival times and deadlines. We extend the problem for the flow over time [5] setting we have, by adding a time dimension to the network, in the spirit of many flow over time problems (e.g. time-repeated graphs). By contrast with the "MCF over time" problem [5], which supports a quite similar definition, BDRT also considers variable active windows for requests, thus not all time intervals are available to all requests. Another difference with [5] is that our time

intervals do not represent any time discretization, but are determined by the job arrival times and deadlines themselves, as in [2]. Let us recall that  $\Omega_T$  is the set of all time intervals obtained, as in [4] by separating the time in at most  $2|T| - 1$  intervals, determined by the at most  $2|T|$  consecutive moments in time represented by the arrival times and deadlines of the  $|T|$  requests. We denote by  $|\omega|$  the duration in time of interval  $\omega \in \Omega_T$ . We next introduce our main optimization variables:  $x_{l,r,\omega}$  is defined as the ratio of the volume  $\nu_r$  of request  $r$  to be transferred during interval  $\omega$  through link  $l$ . We can now write our linear equation system as:

*BDRT* : minimize  $g$

subject to

$$\forall r \in T, \forall n \in N \setminus \{s_r, d_r\}, \forall \omega \in \Omega_G,$$

$$\sum_{e \in O_n} x_{e,r,\omega} - \sum_{e \in I_n} x_{e,r,\omega} = 0 \quad (1)$$

$$\forall r \in T, \sum_{\omega \in \Omega_G} \left( \sum_{e \in O_{s_r}} x_{e,r,\omega} - \sum_{e \in I_{s_r}} x_{e,r,\omega} \right) = 1 \quad (2)$$

$$\forall l \in L, \forall \omega \in \Omega_G, \sum_{r \in T} \nu_r x_{l,r,\omega} \leq g |\omega| cap_l \quad (3)$$

$$\forall r \in T, \forall l \in L, \forall \omega \in \Omega_G \setminus \Omega_r, x_{l,r,\omega} = 0 \quad (4)$$

In the above, for convenience, we defined auxiliary sets  $O_n = \{l \in L : src_l = n\}$  and  $I_n = \{l \in L : dst_l = n\}$  as the sets of outgoing, respectively ingoing edges corresponding to node  $n$ . Equation 1 is the flow conservation constraint pertaining to relay nodes, saying that all data that enters a relay during any time interval, for any request, must leave that node during the same interval. Equation 2 enforces active window-enabled flow conservation at source nodes, asking that the entire volume  $\nu_r$  of request  $r$  is sent from the source during the active window  $\omega_r$ . Equations 1 and 2 render the corresponding flow conservation equation related to destinations superfluous. Equation 3 is the link constraint asking that any link's utilization should be smaller than  $g$ , the overall network congestion, where  $g = \max_{l \in L} (h_l) = \max_{l \in L} (\max_{i \in I} (k_{l,i}))$ . Since  $x_{l,r,\omega}$  is a ratio of volume and  $cap_{l,i}$  is a bandwidth, we multiply the right term by the duration of interval  $\omega$ , under the assumption (proved in [2] and that supports an identical proof for BDRT) that there exists a step-function type solution to the time-domain scheduling.

Let us note that in the current form the system can produce what we call "spontaneous cycles", i.e. optimal solutions whereby a relay spontaneously starts a flow that can cycle back to itself in the network. If this flow does not cause overall congestion  $g$  to exceed its optimal value, these solutions can still be found, since they respect flow conservation, although in practice there will not be any flow originating there. An example

illustrating this is shown in Fig. 1.a). Graph edges are labeled with  $X/Y$ , where  $X$  is the flow scheduled on the edge, and  $Y$  is the edge capacity. Overall network congestion is 0.5 in this case, achieved on edge A-B. As we can see, A-B-D-E-B and A-B-C-B are cycles, however all flow equations are satisfied and congestion is minimal, since extra flow cycles yield congestion below 0.5. This problem is inherent to the fact that our minimization objective is the maximum (and not the sum) of link utilizations and there are multiple optimal solutions. Besides unnecessarily complicating the flow graph, cycles render the optimal schedules useless for practical overlaying. In our example, relay B, although it receives the correct amount of traffic from A, wouldn't know how to correctly forward it and packets will end up roaming for a long time in the cycles. The problem of transforming optimal solutions in cycle-free schedules usable for relaying has not been addressed in either [13], [5], or in other previous related work to our knowledge. To remove cycles, we use an algorithm whereby, for each "instant graph" (what we call the flow graph for a single request and during a single time interval) we iteratively traverse the flow graph from the source in a Depth First Search. Every time we encounter a cycle, we calculate the minimum flow on any of its edges and we reduce the flow on all cycle edges by that value. This will cause at least one edge to be removed, breaking the cycle. The algorithm stops when no more cycles are encountered. Fig. 1.b) shows the cycle-free instant graph corresponding to Fig. 1.a). The resulting graph is still optimal, since link utilizations can only decrease. The complexity of the algorithm is  $O(E^2)$ , where  $E$  is the number of edges, since a DFS takes at most  $E$  steps and there are at most  $E$  edges to be removed in the graph, thus at most  $E$  DFS iterations. In practice, there are only a few cycles produced by our optimizer, since the optimization method we use, simplex, searches the feasible parameter space along its *edges*, thus inherently reducing the number of links used by the optimal solution (non-zero variables in the solution). BDRT enjoys a linear formulation, thus is effectively solvable by well-established methods, for instance simplex or interior point.

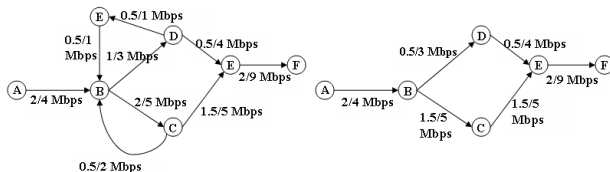


Fig. 1. Left: optimal flow with cycles. Right: equivalent cycle-free flow.

### B. Heuristic routing and oblivious approaches

Oblivious routing algorithms route each packet with no information about routes taken by any other packets

in the network. We quantitatively explore the performance penalty heuristic and oblivious approaches incur against the optimal solution because, not needing any global information, they are much easier to distribute than a global linear optimization. We use a congestion minimizing routing heuristic to independently find paths for each request. We then explore two options: in the first, we feed these paths to an optimal time-domain solver that takes into account all requests (i.e. it is not oblivious); in the second, transfers are scheduled on these paths obliviously to each other, albeit in a congestion-minimizing manner. We detail the 2 approaches next. In previous work[2], one of the present authors has addressed optimal time domain scheduling of bulk transfers for congestion minimization over a set of *predefined* paths and the associated problem, Bulk Data Transfer Scheduling (BDTS), was defined. While BDTS's solution will be optimal for the predefined paths, BDTS does not handle routing, but instead takes  $\Theta_r$ , a finite set of network paths associated with each transfer request as an input. We leverage BDTS by feeding it the paths produced by routing heuristics to get time-domain optimal bandwidth allocation profiles for the respective paths, taking into consideration all requests. We do this to give routing heuristics the best chance against BDRT. In a second approach, each request simply uses the paths, while trying to minimize congestion produced *by its traffic*, obliviously to what other requests exist in the network. An oblivious form of congestion minimization is applied, which can be seen as a generalization of spaghetti scheduling for multipaths. For each request, a simpler linear program is solved, having only one time interval (the request's active window), one request and the visible network limited to the paths found by the routing heuristic (which can still share links). The objective, as before, is to minimize the maximum link utilization produced on the visible links. We evaluate 2 routing heuristics: K Widest Paths (KWP) and K Random Paths (KRP). KWP routes on the K widest (non-disjoint, simple) paths in the network w.r.t. available bandwidth that connect the source and destination. It is a greedy heuristic often used in overlay networks, for instance in the "bandwidth aware routing" proposed in BARON - [8], or in P2P systems (e.g. BitTorrent), where communication is done with preference towards high-bandwidth peers. It is therefore interesting to evaluate KWP's efficiency and effects on network congestion. The second heuristic is KRP, whereby K random paths are picked, in the hope that paths for different requests will not overlap. This is similar to some congestion-avoidant random routing-based strategies, e.g. the ones proposed by Upfal [12]. Since we have 2 heuristics and 2 cases: with/without global time-domain optimization, we have 4 heuristic-based approaches to compare with

the optimal BDRT LP solver : BDTS/KWP, BDTS/KRP, OBLIVIOUS/KWP and OBLIVIOUS/KRP.

### C. The routing overlay

Our scheduler solves the BDRT problem by any of the previously-mentioned algorithms and outputs the solution as a collection of multi-path profiles  $\mu_r(l, i)$ , one per each request. These are currently output as an XML file that contains each request along with profile timesteps, links and corresponding rates. We implement a relaying and traffic shaping overlay, which is a distributed system running on multiple cooperating nodes. On each node, it takes as input an identical XML transfer schedule file, as produced by a global scheduler. Based on it, each node independently deduces its role(s) within requests, nodes with which it needs to communicate and the respective time-rate profiles for each link. For any given request, nodes can have the role of either sender, receiver or relay; for various requests, a same node may have multiple roles. Relays forward traffic sent by senders or other relays to receivers or other relays, according to profile. Senders and relays limit their sending rates variably in time according to the profile. Relay-receiver, relay-relay, sender-relay and sender-receiver synchronization is implicit, that is relays and receivers consider as the start moment of the schedule the moment they receive the first packet from their source, then rely on the local timer to follow the profile; senders are initially synchronized by a central Synchronizer node, that sends a START\_SCHEDULE packet to all senders as it receives news that all connections have been made; these 2 mechanisms remove the need for synchronized clocks on the nodes, which is non-trivial.

### D. Dealing with cross traffic

Our optimizer targets overlays running over wide area public networks, where cross-traffic is present and available bandwidth on links is variable. To address resource variability, rather than having single values for the per-link available bandwidths, we maintain distributions of values, representing available bandwidth values historically observed on overlay links. To build these distributions, we use active measurements with a certain frequency obtained with the `pathchirp` tool. For PlanetLab, as discussed in section IV, public measurement traces produced within the S3 project [14] are leveraged. To generate topologies to feed to our algorithms, we use these distributions to derive likely expected values. Various strategies are possible, including considering the latest value, the minimum, average, median, values observed around the same times of the day as the transfer needs to take place etc. We found that it is most effective to use a certain percentile of the distribution as an expected value. We characterize each distribution by a Complementary Cumulative Distribution Function

(CCDF), which gives us the probability  $P(X)$  that the available bandwidth will be greater than a certain value  $X$ . By fixing a certain overall probability margin  $m$  that we are willing to accept (for instance 90%), we derive, for each link, the respective bandwidth value  $X$  such that  $CCDF(X) = m$ , representing the value over which the available bandwidth on that link is likely to be, with probability  $m$ . In our tuning, we used  $m = 0.9$ , since we found the distributions of available bandwidths on both Intrigger and Planetlab to have a very high variance, thus using a median proved to be too optimistic for accurate prediction.

## IV. EVALUATION

We evaluate our system by both trace-driven simulations and real-world measurements on Planetlab and the Intrigger wide area cluster.

### A. Simulations

We do trace-driven simulations based on Planetlab public traces from the S3 measurement project, spanning about 7 months. From these traces, we select `pathchirp` available bandwidth probes and we derive expected values from these distributions. We then build (almost) full mesh networks, containing  $N$  randomly picked nodes, as well as all the links available in the S3 traces that directly connect these nodes. Further, for some evaluation setups, we generate overlays that are increasingly well connected rather than full meshes, to estimate the impact of the number of alternate paths in the overlay on our scheduling algorithms. To obtain increasingly well connected networks, we start from a full mesh and generate Erdos-Renyi  $G(n, p)$  random networks [3], with variable values of the connection probability  $p$ . E-R graphs are proved to be connected w.h.p. if the per-node connection probability  $p$  is chosen above a threshold of  $(\ln(N)/N)$ , where  $N$  is the number of nodes. We variate  $p$  from this minimal connectivity value up to 1, and thereby obtain networks in which there are increasingly many alternate paths for any given request, since the distribution of node degrees in E-R networks is uniform. Besides network connectivity (subsection IV-A2), other parameters we variate are  $K$ , the number of alternate paths that the KWP and KRP heuristics look for (IV-A1) and the number of requests in the network (IV-A3).

1) *Variable K parameter:* For a network of 40 nodes and 10 requests, we use either a slightly connected overlay (connection probability  $p = 0.32$ ) or a full mesh ( $p = 1$ ) and we variate parameter  $K$  (5,10,20,30,50,70). For each case, we generate 3 random problems, each having 10 randomly generated requests: random source and destination, random arrival times within a given window of 50 minutes, active window of 16 minutes and randomly distributed volume sizes (between 10MB and

50MB). We note that BDRT performs much better than the others on the full mesh, as it seems to effectively take advantage of high path diversity, and still has a smaller advantage on the barely connected network. Heuristic algorithms do better, but are also slower as  $K$  increases. Figures 2 and 3 plot the congestion factor achieved by all algorithms on the two networks. Oblivious approaches achieve congestions at least an order of magnitude worse than their non-oblivious counterparts, therefore in both figures, graph b) is a zoomed-in version of a), only keeping the more effective non-oblivious algorithms. BDTS-based algorithms improve as  $K$  grows but remain suboptimal. Also, KRP seems to be the better strategy in high connectivity networks, as KWP does better at low connectivity. We believe this is because when there are more alternate paths, KWP biases towards the same few "fat" paths to well-connected relays, thus increasing congestion on them, whereas KRP does not. On the other hand, when there are few alternate paths, KWP wins by choosing wider paths than KRP's random choice. Figures 4a) and b) show the algorithm runtimes in the same setup. In the current implementation, we use a sequential linear solver based on GLPK's standard simplex implementation, with some performance tuning. The configuration we ran on is Dual Core AMD Opteron 2.4 GHz, 4GB RAM. Parallelizing the optimizer or using other methods, such as interior point, might considerably speed up the optimization. We see that runtimes increase along with  $K$  and that the full mesh takes longer to solve than the low connectivity network. BDRT is slower up to the point where  $K=70$  (low connectivity), respectively  $K=50$  (full mesh), at which point BDTS/\* become comparable or slower. The heuristic strategies are considerably faster, since the linear system to solve is much simpler than BDRT and BDTS. OBLIVIOUS/KRP scales best in runtime as  $K$  grows, followed by OBLIVIOUS/KWP that scales worse at low connectivity. Comparing congestion factor and runtime graphs, we can argue that heuristic strategies can in principle get close to BDRT for high values of  $K$  and if aided by BDTS time-domain optimization (although BDRT still provides a factor of 2 improvement in the high connectivity case), but at these high  $K$  values their runtimes become similar or slower than BDRT as well.

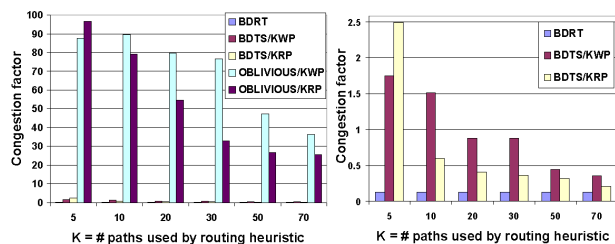


Fig. 2. High connectivity, variable  $K$ , congestion factor: a) All algorithms; b) Non-oblivious algs.

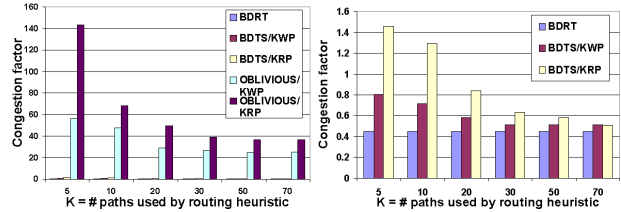


Fig. 3. Low connectivity, variable  $K$ , congestion factor: a) All algorithms; b) Non-oblivious algs.

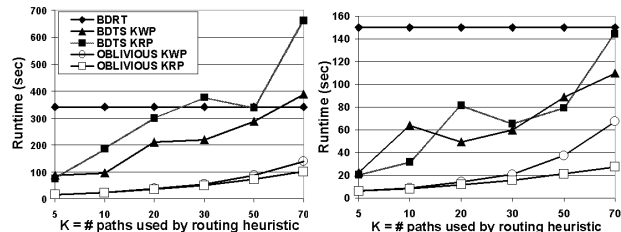


Fig. 4. Variable  $K$ , all algorithms - runtime. a) Full connectivity; b) Low connectivity random network.

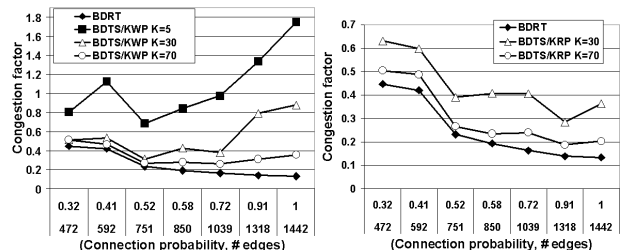


Fig. 5. Influence of variable connectivity on congestion factor. Optimal BDRT vs. a) BDTS/KWP,  $K=5,30,70$ ; b) BDTS/KRP,  $K=30,70$ .

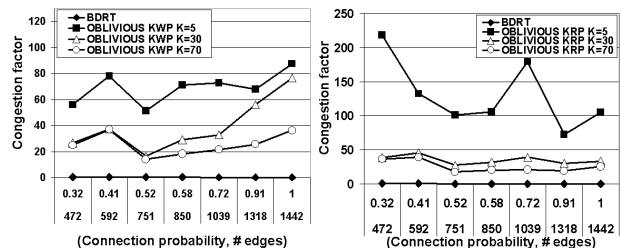


Fig. 6. Congestion factor =  $f(\text{variable connectivity})$ . BDRT vs. a) OBLIVIOUS/KWP; b) OBLIVIOUS/KRP;  $K=5,30,70$ .

2) *Variable network connectivity*: For a network of 40 nodes and 10 requests, we variate the overlay connectivity from slightly above minimal ( $p = 0.32$ ) to full mesh ( $p = 1$ ). Intermediate points are  $p=0.41,0.52,0.58,0.72,0.91$ . For the heuristics, we use a few values of  $K$  (5, 30 and 70). Request generation is as before. Results show that, as connectivity increases, BDRT does better, whereas the others do worse, especially for small  $K$ s. Also, all algorithms get slower as connectivity increases. Figures 5a) and b) and 6a) and b), plot the congestion factor achieved by BDTS/KWP, BDTS/KRP, OBLIVIOUS/KWP and OBLIVIOUS/KRP against BDRT, respectively, when varying connectivity. As in all experiments, algorithms are given identical sets

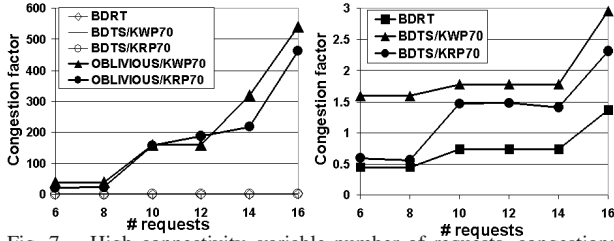


Fig. 7. High connectivity, variable number of requests, congestion: a) all algorithms; b) zoomed in, non-oblivious algorithms

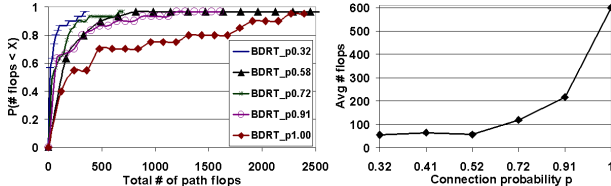


Fig. 8. Variable connectivity, average number of path flops during transfer by the optimal strategy. a) CDFs b) average.

of problems to solve. In Fig. 5b), we left out BDTS/KRP,  $K=5$  because it performed much worse compared to the others (congestion factors around 3) and caused the graph to zoom out. We clearly see that BDRT improves as the network has more alternate paths. Heuristic algorithms generally get worse as the network gets more connected (except for KRP  $K=70$ , which scales best among all heuristics). \*/KRP algorithms generally scale better than \*/KWP w.r.t. connectivity. All algorithms remain sub-optimal, with oblivious ones faring much worse than non-oblivious. Graphs with algorithm runtimes are omitted for brevity, but as before, show that runtimes increase with the number of links and BDRT is comparable to BDTS/\* and slower than OBLIVIOUS/\*.

3) *Scaling in number of requests:* Figures 7 a) and b) show the congestion achieved by various algorithms as increasingly many random requests are added to the problem. Fig. b) is a zoomed-in version of a), excluding the worst-performing oblivious algorithms. We used a 30-node Planetlab full connectivity network, and varied the number of requests. To ensure that the network is getting increasingly congested, we added at each step new random requests on top of the previous ones. Requests have random sizes of at most 100 MB and active windows of 100 seconds. We see that BDRT scales best in number of requests, followed by BDTS/KRP and BDTS/KWP ( $K$  was 70 for both).

4) *Insights on routing and time domain scheduling:* To gain insight into what helps BDRT achieve optimal solutions, we evaluate several metrics such as number of paths used in parallel, number of hops paths have, number of path flops (path switches from a time interval to the next) and percentage of active window used in sending. We find that BDRT mainly achieves low congestions by using more network paths and flopping

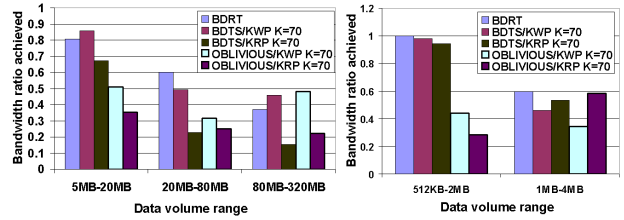


Fig. 9. Ratio of scheduled bandwidth achieved at different network congestion levels: a)Intriggrer b) Planetlab

these paths more often. Figures 8 a) and b) show CDFs, respectively average values for the number of path flops performed by BDRT. Averages are quite high, from 55 to 601 and increasing with network connectivity, as compared to BDTS (e.g. 1.4 to 3.1 for  $K=5$ ), which tends to do less flopping as connectivity increases, due to it sticking to the same available wide paths. Similar conclusions can be drawn for paths used in parallel, but graphs are omitted here for brevity. In a 40-nodes network, BDRT uses on average between 24 distinct parallel paths (low connectivity) to 204 (full mesh). Heuristic algorithms are upper bounded in this respect by  $K$  and also tend to use less than  $K$  paths. We believe diverse path usage to be the main advantage of BDRT, allowing it to exploit network parallelism at high connectivities. On the other hand, average number of path hops is 6.4 for BDRT and 2.4 for BDTS/KWP. Regarding time domain metrics, algorithms are quite similar, with the note that, by the nature of spaghetti scheduling, OBLIVIOUS algorithms use 100% of the active window, whereas BDRT (74%) and BDTS (77%) tend to delay starts and/or finish earlier to accommodate other requests. Shorter paths and full window usage are some advantages of heuristic, respectively oblivious approaches over BDRT to address in future work.

### B. Real-world transfers

We use our traffic shaping overlay to evaluate the various algorithms in real world transfers. In practice, whereas scheduling algorithms minimize the congestion they produce in the network, we found it good practice to use a slightly higher rate limit value than computed by the scheduler, uniformly for all requests. This is due to the fact that although it is more likely to get a lower bandwidth than a high one, bandwidth is highly dynamic in reality, thus even for small values there can be fluctuations. Using a safety margin (tuned in practice to 10 to 20% of the target value, depending on the testbed) helps compensate for this variability. To evaluate the algorithms, we define a metric called *bandwidth ratio achieved*. A request's transfer schedule is composed of a set of scheduled transmission rates that, if fully achieved, ensure that the deadline is made. We define the bandwidth ratio achieved as the ratio between actual bandwidth achieved for the request in the

real transfer and the theoretically scheduled transmission rate, according to profile. We use this metric instead of characterizing the distribution of missed deadlines because we found it to be more robust, less jittery, less heavy tailed and it characterizes well the algorithms' ability to realistically pick appropriate links. Figure 9 shows average values of bandwidth ratios achieved by algorithms for various network utilization levels. Figure a) refers to the Intrigger wide area cluster and b) to Planetlab. To variate the bandwidth required from the network, we generate sets of requests of fixed active window of 100 seconds and volumes uniformly distributed in a certain interval that we variate. For Intrigger, we used 3 intervals: 5-20,20-80 and 80-320MB, roughly corresponding to bandwidth requirements ranging from 0.4-27 Mbps *per request*. We generate sets of 20 requests per problem, using 12 available Intrigger WAN front-end nodes. We see that for this cluster, BDTS/KWP is a strong competitor for BDRT, even surpassing it for certain problem sizes. Same goes for OBLIVIOUS/KWP at high loads, which suggests that Intrigger has at least some well-provisioned paths that are hard to self-induce congestion onto, thus suit KWP well. KRP/\* algorithms perform poorly especially at high loads. On Planetlab, we use data volume ranges of 512KB-2MB and 1MB-4MB, corresponding to bandwidth requirements of 42 to 335 Kbps per request. We use sets of 20 requests and overlay sizes of 20 nodes. On the lower bandwidth setup, we find that all time-optimized algorithms perform much better than oblivious ones and BDRT performs the best. For the more overbooked network case, BDRT remains the best, but KRP-based algorithms get quite close to it, while KWP loses effectiveness. Overall, KWP fared better on the cluster, whereas KRP was better in the widely distributed, less over-provisioned, flakier network. Compared to congestion factor results, the conclusion is that, whereas real networks seem to tolerate congestion to some degree, self induced congestion indeed becomes a factor at non-trivial bandwidth requirements, so a global optimization makes sense.

## V. RELATED WORK

We already discussed in Sections II and III the positioning and differences between our own work and seminal efforts such as the theoretical work on flow over time problems by Hall et. al [5], Wang's traffic engineering work [13], networks with advanced reservations [4] as well as the work of one of the current authors on BDTS and time-domain congestion minimization [2]. Regarding the theoretical optimality of oblivious algorithms, Racke found [10] that, for generic graphs, there exist oblivious routing algorithms that achieve congestion a polylogarithmic factor ( $O(\log^3(n))$ ) worse than optimal. Later, Harrelson et al. [6] have proposed a polynomial-

time oblivious algorithm that achieves poly-logarithmic competitiveness. To date, we have not found an evaluation of oblivious algorithms against an optimal solution through either simulations or experiments, at least not in the context of actual Internet overlays. By tackling the overlay routing problem, we also relate to a plethora of work on routing overlay networks, such as [1][11][7]. Space limits do not allow for an overview of this area. However, to the best of our knowledge, we make the first attempt to propose the global optimization of upcoming bulk transfers in overlays among cooperating nodes so as to cope with cross-traffic.

## VI. CONCLUSIONS AND FUTURE WORK

We propose a global scheduler for bulk data transfers, targeted at applications using overlays or doing application-level routing over public networks. Evaluation shows that indeed BDRT produces optimal schedules, outperforming competitor approaches, which also reflects in actual transfer performance. Future work includes looking at native-layer, rather than overlay-layer links for congestion minimization, distributing the optimizer so as to speed up scheduling decisions, finding optimal schedules with improved practical characteristics (shorter paths, full active window utilization), as well as developing hybrid strategies trading strict congestion optimality for more greedy approaches.

## REFERENCES

- [1] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.
- [2] Binbin Chen and Pascale Vicat-Blanc Primet. Scheduling bulk data transfers in grid networks. In *CCGRID*, 2007.
- [3] P. Erdos and A. Renyi. On random graphs i. In *Publicationes Mathematicae*, 1959.
- [4] R. Guerin and A. Orda. Networks with advance reservations: The routing perspective. In *IEEE INFOCOM*, 2000.
- [5] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: efficient algorithms and complexity. In *Journal of Theoretical Computer Science*, volume 379, 2007.
- [6] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.
- [7] P. Karbhari, M. Ammar, and E. Zegura. Optimizing end-to-end throughput for data transfers on an overlay-tcp path. In *IFIP Networking Conference*, 2005.
- [8] Sung-Ju Lee, Sujata Banerjee, Puneet Sharma, Praveen Yalagandula, and Sujoy Basu. Bandwidth-aware routing in overlay networks. In *INFOCOM*, 2008.
- [9] C. Martel. Preemptive scheduling with release times, deadlines, and due times. In *Journal of ACM*, volume 29, 1982.
- [10] H. Racke. Minimizing congestion in general networks. In *FOCS* 43, 2002.
- [11] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. Overqos: An overlay based architecture for enhancing internet qos. In *NSDI*, 2004.
- [12] E. Upfal. Efficient schemes for parallel communication. In *ACM Journal*, volume 31, 1984.
- [13] Y. Wang and Z. Wang. Explicit routing algorithms for internet traffic engineering. In *ICCN*, 1999.
- [14] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee. S3: a scalable sensing service for monitoring large networked systems. In *SIGCOMM workshop on Internet network management*, 2006.