Optimizing deadline-driven bulk data transfers in overlay networks

Andrei Agapi* Sebastien Soudan+ Marcelo Pasin$ Pascale Vicat-Blanc Primet+
Thilo Kielmann*

Vrije Universiteit Amsterdam, The Netherlands*
Ecole Normale Superieure Lyon, France+
Universidade de Lisboa, Portugal$

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 (i.e. minimizing maximum link utilization). The challenge in this optimization is to jointly address routing of transfers within the overlay and time-domain transfer scheduling. We formally define and address the associated problem, the Bulk Data Routing and Transfer(BDRT). We present a linear programming-based solution to BDRT, 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 simulations and Internet transfer experiments. Our trace-driven simulations leverage PlanetLab public traces collected by the S3 measurement project, spanning about seven months. Measurement-based experiments are performed 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.

1 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, which is unable to provide QoS guarantees for applications using it, systems such as opportunistic routing overlays [5] [18] [12] 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 needed from the underlying network. Such overlays typically work by routing around congestions and network bottlenecks in an attempt to improve QoS. Besides routing overlays, many other overlays and p2p systems, such as Bittorrent [1] or Tribler [15], 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 especially from the point of view of novel emerging applications, such as data intensive p2p computing, cloud computing [3], distributed stream processing [8] and, in general, for any distributed application in which distant peers have common, rather than selfish goals. Other examples include commercial distributed organizations that need to transfer and process, in real-time, data produced at geographically distributed locations or soft real-time, SETI@home-like processing of large scale data, as produced by emerging systems such as the CERN LHC or new-generation distributed radio telescopes [2] [4]. In these 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 and/or may need to be synchronized. For such applications, cooperatively transfering and processing the data in a globally optimal manner, such that all of the deadlines are met, is a much more appropriate approach then acting as a collection of selfish peers. Even in the case of applications where peers have selfish goals (such as file downloads), using a global optimization might, in some instances, yield better results than the current approaches, whereby each peer generally uses the system in a greedy/selfish manner.

We do 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 unpredictable 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 we *require* from network links. A desirable effect of minimizing the required bandwidth 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 congestion minimization approach maximizes the probability that *all* transfer deadlines are actually met. This is one of the main reasons we focus on minimizing network congestion as a global optimization objective in this work.

Basically, the problem we address, Bulk Data Routing and Transfer (BDRT), is to jointly optimize the routing of the data 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 at overlay link (rather than native link) level, i.e. we minimize congestion on overlay links.

This paper is structured as follows. Section 2 defines the model and formulates the BDRT problem we are trying to solve. We describe our solution in section 3. In fact, this includes an optimal, linear programming-based approach, as well as a few approaches based on heuristic routing. In section 4 we describe the routing and traffic shaping overlay infrastructure we developed. Section 5 describes our experimental setup and evaluation results. Section 6 addresses related work and Section 7 outlines future work, as well as possible applications of our system.

2 PARADIGM AND PROBLEM DEFINITION

The paradigm we work in is the one previously defined in work such as [9] or [7]. In this paradigm, bulk transfers are characterized, besides the data volume ν_r to be transmitted, by an arrival time η_r (at which data is available at the source) and a deadline ϕ_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 is known or 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 decribed as tuples (ν_r, ω_r), i.e. transfer volume and active window, find the network paths to route each transfer on, as well as the bandwidth allocation profile λ_r for each of these requests and each of these paths such that all deadlines are met and the overall network congestion factor is minimized. The network congestion factor is defined as the maximum link utilization over all links in the network, over all intervals.

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. [7] 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 [14]), the problem defined posseses an optimal solution with the bandwith 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 bandwith is limited, for any task, is a constant.

In the current work, we show that BDRT, i.e the problem of jointly optimizing routing and time domain scheduling is in P and effectively solvable by linear programming (e.g. by the simplex or interior point methods). The solution to BDRT is in the form of a so-called multipath profile, that is 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 at each time interval as well as the rate at which request r 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 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 λ on each interval ω_i.[1] Profile intervals $\omega_i \in \Omega$ are defined by endpoints η and ψ, as $[\eta_i, \psi_i]$, where $\forall i, \eta_i < \psi_i$ and $\psi_i = \eta_{i+1} | 1 \leq i < |\Omega|$.

[1] Step functions are comonly defined as $\lambda(t) = \sum_{i=0}^{n} \alpha_i \chi_{\omega_i}(t)$, where $n \geq 0, \alpha_i$ are real numbers, ω_i are intervals, and $\chi_\omega(t)$ is the indicator function that yields 1 or 0 depending on the membership of t to ω.

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. A link is a directed edge on G, defined as $l = \langle s_l, d_l, c_{l,\omega} \rangle \in L$, characterized by a source node s_l, a destination node d_l, and a capacity cap_l.

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 4.2).

Definition 3. A request $r = \langle s_r, d_r, \nu_r, \eta_r, \phi_r \rangle \in N \times N \times \mathrm{R} \times subsets(I)$, is defined by a source node s_r, a destination node d_r, a volume to transfer ν_r, an arrival time η_r, and a deadline ϕ_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, Ω_r is the set of those time intervals (as defined by the divison 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 requests T is defined as $\Omega_T = \bigcup_{r \in T} \Omega_r$. Ω_T contains a finite number of intervals.[2]

Definition 5. For a link $l \in L$ and time interval $i \in I$, we define $k_{l,i} \in \mathrm{R}+$ 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 refered 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 $g \leq 1$, that is 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.

Regarding the complexity of BDRT, refering to [10], where an extensive analysis of various flow over time problems is presented, we find that can regard BDRT as an instance of the "min-cost multicommodity flow over time with uniform pathlengths" problem. Theorem 1 of this paper proves this problem to be in P. This indeed 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, which is indeed the case for long-lived data transfers over Internet. By this we do not mean that latency does not influence throughput (e.g. by longer RTT of ACK packets in TCP that can cause longer retransimission times of lost packets), but that the "transit time on pipes", as defined in [10] can be considered negligible in the context of long-lived Internet data transfers.

In the above-mentioned reduction, one has to note that indeed, while we consider transit times negligible (thus uniform)

[2] In practice, the actual union function used is not the traditional one. The union is given by the set of intervals $\omega = [\sigma_i, \sigma_{i+1}]$ such as σ_i is the i-th element of the ordered set of endpoints η and ϕ of the interval sets being united.

in BDRT, the MCF problem does not consider requests with different start times and deadlines. However, a reduction can be straightforwardly achieved by adding two extra graph vertices per each request, with transit times corresponding to the start time and deadline, respectively, to the source/destination of each request, which still keeps path lengths uniform with respect to transit times. Our problem can thus be seen as an instance of "min-cost multicommodity flow over time with uniform path-lengths" of polynomial size, as there are only two vertices added per each request. Furthermore, as will be seen in the next section, BDRT can be written in a (pure) linear form, thus making it effectively solvable in practice.

As an aside, [10] shows that the more general "min-cost multicommodity flow over time" problem, that considers both link capacities and transit times, is strongly NP-hard, but the demonstration relies exactly on path non-uniformity w.r.t. transit times.

In the next section, we describe the solutions we propose for the BDRT problem.

3 Solution description

3.1 The linear programming approach

We find that the problem can be written in a LP form, inspired by Wang's work in [19]. Done in the context of Internet core traffic engineering, [19] solves the routing problem for the case in which transfer requests arrive simultaneously and there are no deadlines for transfers and no data volumes characterizing transfer requests, but rather each request is treated as a continuous bandwidth reservation, typical for Internet core flows. Requests thus have variable bandwidth demands, but no specific arrival time and deadline, and are basically infinite flows. Analogously, we propose a linear optimization method for multipath routing for the BDRT problem, in which requests are characterized by variable data volumes, arrival times and deadlines and there is an extra time dimension. We extend the problem and its solution for the flow over time [10] setting we have, that is we basically add a time dimension to the network, in the spirit of many flow over time problems (e.g. time-repeated graphs). Unlike the "min cost multi-commodity flow over time" problem, defined in [10], which supports a quite similar definition, our equation system also supports various arrival times and deadlines (active windows) for requests, thus, by contrast with the classical flow over time setting, not all time intervals are available to all requests. Also, as we will see further, unlike in [10], our time intervals do not represent any time discretization, but rather are determined by the job arrival times and deadlines themselves, as in [7].

Let us recall that Ω_T is the set of all time intervals obtained, as in [9] 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. Let us denote by $\|\omega\|$ the length (i.e. duration in time) of interval $\omega \in \Omega_T$. To define the equation system, let us denote our main optimization variables:

Definition 6. $x_{l,r,\omega}$ is defined as the ratio of the volume ν_r of request r to be transfered during interval ω through link l.

Finally, for convenience, we define the auxiliary sets $O_n = \{l \in L : src_l = n\}$ and $I_n = \{l \in L : dst_l = n\}$ to represent the sets of outgoing, respectively ingoing edges corresponding to node n. Now, with the above notations, we can write our linear equation system as:

$BDRT$: minimize g

subject to

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

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

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

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

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

In the system, equation 1 is a 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. Let us note that this corresponds to the "no storage at intermediate nodes" case in multicommodity flow problems. The storage case can be easily implemented by summing equation 1 over all intervals, rather than requiring it to be true for any interval. However, we do not focus on this case in the current paper. Equation 2 enforces flow conservation at source nodes, asking that the entire volume ν_r of request r is sent from the source during the active window ω_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. Since $g = \max_{l \in L}(h_l) = \max_{l \in L}(\max_{i \in I}(k_{l,i}))$, we can safely exclude $k_{l,i}$ and h_l from the system and only use the variable g, since the objective is to minimize it. Since $x_{l,r,\omega}$ is a ratio of volume, whereas $cap_{l,i}$ is a bandwidth, we multiply the right term by the duration of interval ω, under the assumption (proved in [7]) that there exists a step-function type solution to the problem, in the form of constant bandwidths in each time interval. The last constraint is strictly necesary only for those $l \in O_{s_r}$, i.e. the source's outgoing edges, from the point of view of finding the optimal solution. However, we define it for all links, so as to find cleaner, more understandable solutions, that are easier to use in practical overlaying. Besides helping the solver run faster by explicitly zeroing out rates outside the requests' active windows, this constraint helps avoid what we call "spontaneous cycles", that is optimal solutions whereby a relay spontaneously starts a flow that can cycle back to him in the network. If the flow is still below the minimum overall congestion g, these solutions can still be returned by the optimizer, since they are correct, although in practice there will not be any flow originating there. So although theoretically not strictly necessary, equation 4 is written that way to simplify schedules, for practical use by our overlaying infrastructure.

Let us note that flow cycles can still be produced by the optimizer even in the current form, if the extra flow they introduce does not cause overall congestion to exceed its optimal value. An example illustrating this is shown in Fig. 1.a). The graph edges are labeled with X/Y, where X is the flow scheduled on th eedge, and Y is the edge capacity. The 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 the flow equations are satisfied and the congestion is still minimal (extra flow in cycles yields congestion below 0.5), therefore the solution is valid and optimal. This problem is inher-

ent to the fact that our minimization objective is the maximum (and not the sum) of link utilizations in the network and there are multiple optimal solutions. However, maximum utilization is indeed the correct objective needed to maximize chances that deadlines are made, as explained. Besides unnecessarily complicating the flow graph, cycles actually render the optimal schedules useless for practical overlaying. In our example for instance, relay B, although it receives the correct amount of traffic from A, wouldn't know how to forward it correctly 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 [19], [10], or in other previous related work to our knowledge.

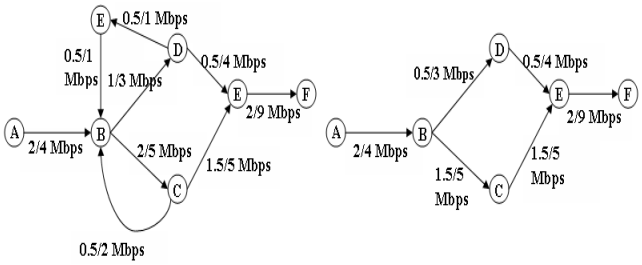

Figure 1: a) Optimal flow with cycles. b) Equivalent cycle-free flow.

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 edge of the cycle 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. Useless vertices are removed as well. The algorithm stops when no more cycles are encountered. Fig. 1.b) shows the cycle-free instant graph corresponding to Fig. 1.a). The order in which cycles are removed is not important for the result correctness and the resulting graph will still be optimal, since link utilizations can only decrease. The complexity of the algorithm is $O(E^2)$, where E is the number of egdes, 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 however, there are only a few cycles produced by our optimizer, since the optimization method we use, simplex, unlike interior point methods, performs the search along the *edges* of the feasible parameter space, thus inherently reducing the number of links used by the optimal solution (the non-zero variables in the optimal solution). In practice the runtime of the cycle removal step is quite small and negligible compared to the linear optimization, although it could still be improved by using more efficient cycle removal algorithms.

BDRT enjoys a linear formulation, thus is efectively solvable by well-established methods, for instance simplex or interior point method solvers. This is true for the "pure multipath" case (that is, the case when requests are splittable over many paths in the network and there is no upper bound on the number of network paths any transfer can use). In the single path or at-most-K-paths cases, the problem becomes an Integer Linear Programming (ILP) one, and we believe it to be NP, equivalent to the K-disjoint paths problem, similar to the more particular respective problem discussed in [19]. However, as for the simpler case, there might exist efficient heuristics for these ILP problems, for instance based on rerouting flows starting from the optimal multipath solution. Finding such heuristics is an interesting problem by itself and is subject of future work.

3.2 Heuristic routing and oblivious approaches

Oblivious routing algorithms are algorithms that route each packet with no information about the routes taken by any other packets in the network. Because they need no global information, they can be easily distributed and are quite commonly used. The reason we explore heuristic and oblivious approaches is that, although they are likely to yield sub-optimal schedules (actually, oblivious routing algorithms have been proven to be suboptimal in [16]), we find it interesting to compare them against the optimal solution. If for no other reason, because, albeit suboptimal, they are much easier to distribute than a global linear optimization therefore we find it interesting to quantitatively explore the performance penalty they incur in terms of congestion achieved.

In this approach, we use an oblivious routing heuristic that attempts to minimize congestion 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 when performing the optimization (i.e. it is not oblivious); in the second, transfers are scheduled on these paths obliviously to eachother, albeit in a congestion-minimizing manner. We detail the 2 approaches next.

In previous work [7], one of the present authors has addressed optimal time domain scheduling of bulk transfers for congestion minimization over a set of *predefined* paths. The associated problem, Bulk Data Transfer Scheduling (BDTS) was defined, proved to be polynomial, and a linear programing-based global optimization solution for it was proposed. Solving BDTS can globally optimize the scheduling in the time domain of a set of bulk data transfers over a *predefined* set of paths in a network. While the solution will be optimal for the given paths, BDTS does not handle routing, but instead takes the network routes that transfers should use as an input. Therefore, BDTS also takes as a problem input Θ_r, a finite set of network paths associated with each transfer request, representing the paths the request can use to transfer data, as found by a separate algorithm. Bulk transfer requests are thus defined as a triples $(\nu_r, \omega_r, \Theta_r)$, where members mean the associated data volume, active window (arrival time to deadline) and set of paths, respectively. The optimal solution produced is in the form of bandwidth allocation profile λ_r for each request r and its associated paths, such that all deadlines are met and overall network congestion factor is minimized. For more details on BDTS, please see [7]. We leverage BDTS by feeding it the paths produced by routing heuristics to get optimal bandwidth allocation profiles in the time domain for the respective paths, taking into consideration all requests. We do this to give the routing heuristics the best chance against BDRT, by optimizing time-domain allocation. Let us note that in this case global information on all transfer requests is still needed, i.e. the approach is only oblivious wrt routing, but not to time-domain optimization.

In the second approach, each request simply uses the paths, while trying to minimize congestion produced by itself, obliviously to the other requests. An oblivious form of congestion minimization is applied, which can be seen as a generalization of spaghetti scheduling for multipaths. Basically, for each request, a simpler linear program is being solved, whereby there is only one time interval (equal to the request's active window) and the visible network is only 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 on

the visible links. Basically, each sender minimizes the congestion produced *by its traffic*, obliviously to what other requests exist in the network.

We experimented 2 routing heuristics: K Widest Paths (KWP) and K Random Paths (KRP). In short, KWP routes on the K widest paths in the network wrt available bandwidth that connect the source and destination. It is a greedy heuristic often used by or proposed in overlay networks. For instance it is a generalization of the "bandwidth aware routing" proposed in BARON - [13], but in many other P2P systems (including for instance BitTorrent) communication is done with preference towards and downloads are more likely to be from high-bandwidth peers. Therefore, it is interesting to evaluate KWP's efficiency and its effects on network congestion. The second heuristic we use is KRP, whereby K random paths are chosen to route on, in the hope that paths for different requests will not overlap. This is similar to many congestion-avoidant routing strategies, for instance the random routing-based ones, previosly proposed by Upfal [**?**].

We are aware that there exist in literature more sophisticated congestion minimization oblivious routing algorithms than KWP or KRP, for instance Racke's work on network tree decompositions [16] and later developments [11] [6]. However, besides the fact that in practice real-world overlays tend to use much simpler solutions, the focus of the current paper was not on oblivious routing, but rather we aimed to compare the optimal algorithm with some representative, conceptually simple, heuristic approaches and gain some insight into where and how performance gains can be achieved. For this purpose, we found KWP and KRP more instrumental than more complicated oblivious algorithms.

Since we have 2 heuristics and 2 cases: with/without global time-domain optimization, we have 4 heuristic-based approaches : BDTS/KWP, BDTS/KRP, OBLIVIOUS/KWP and OBLIVIOUS/KRP. We will compare all these with "BDRT"=the optimal LP solver.

4 DATA TRANSFER IMPLEMENTATION

4.1 The routing overlay

Our scheduler solves the BDRT problem by any of the previously-mentioned algorithms and outputs the solution in the form of a collection of multi-path profiles $\mu_r(l, i)$, one per each request. Currently these are output as an XML file that contains each request along with the timesteps and links, as well as the corresponding rates. Currently, internally, our scheduler uses the glpk linear solver package to solve the various equation systems by the simplex method.

We implemented Traffic Shaping Overlay Network (T-SON) a relaying and traffic shaping overlay, which has the following characteristics:

- It is a distributed system running on multiple cooperating nodes
- On each node, it takes as input the same XML transfer schedule file as produced by our scheduler
- Based on this file, each node independently deduces its roles within each request, with which nodes it needs to comunicate and the respective time-rate profiles for each link; identification of nodes in schedules is done by either hostname or IP
- Transfer schedules are distributedly run on the underlying testbed (we deployed it on PlanetLab and Intrigger, but it can run almost anywhere, being implemented in pure Java)
- 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; senders are synchronized by a separate, 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; desynchronizations could happen in the order of at most hundreds of milliseconds, reflecting 1-way latency differences between the Synchronizer and various senders, which we believe is acceptable in the context of transfers taking at least minutes and reasonably long profile timesteps
- For a certain request, nodes can have the role of either sender, receiver or relay; of course, for different requests in the schedule a same node may have multiple roles, so multiple servers may start on each node (only 1 server for each of relay, sender, and receiver roles is needed for an entire run, running on the same ports, since each server can process multiple requests, because requests first identify themselves to the server by sending an initial ID packet)
- Relays forward traffic sent by senders or other relays to receivers or other relays, according to profile
- Senders and relays limit the sending rate variably in time according to the profile
- All nodes output statistics on how long the actual transfers took, actual achieved bandwidths per interval, milliseconds by which deadlines were missed etc
- Rate limitation is done based on the technique from [**?**], relying on thread sleeping after each packet sent and compensation of overslept times. We found our implementation quite accurate, especially since the underlying testbeds we tested provide nano-second system timer precision

We believe this overlay infrastructure can be of general purpose use, as long as a certain scheduling algorithm is "plugged in", by outputing well-formated schedule description files. In this paper, we used this infrastructure to estimate the efficiency of the previously-mentioned scheduling algorithms. Specifically, we measured the distribution of times by which deadlines are missed for transfer jobs, as well as the ratio between the actual achieved bandwidth and the scheduled bandwidths on links for each of the algorithms. Evaluation results are presented in the next section.

4.2 Dealing with cross traffic

Our optimizer targets overlays that might be running over wide area, public networks, where cross-traffic is present and avaiable bandwidth on links is variable. As a first step, we consider overlay-level links. Extending the system to native layer links is subject of future work.

To address resource variability, rather than having single values for the per-link available bandwidths, we maintain distributions of values, which represent available bandwidth values historically observed on overlay-level links. To monitor the links and build these distributions, we currently use

active measurements with a certain frequency. Individual probes are available bandwidth measurements obtained with the pathchirp [17] tool. For PlanetLab for instance, as discussed in section 5, public measurement traces produced within the S3 project [20] can be leveraged.

To find appropriate paths and transfer schedules by 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. However, we found that it is most effective to use a different strategy, whereby we 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 on a given link will be greater than a certain value X. Then, by fixing a certain overall probability margin m that we are willing to accept (for instance 95%), we derive, for each link, the respective bandwidth value X such that $CCDF(X) = m$. This basically gives us an expected value over which the available bandwidth on that link is likely to be, with probability m. These likely expected values are then used as link weights by our global optimization algorithms. We found in practice that, to achieve reliable results it is best to use quite conservative values for probability m, e.g. 90%. We denote the corresponding bandwidth value, e.g. the value smaller than 90% of the values observed in the distribution (10-th percentile), by $ccdf\_cv_m$, for *CCDF conservative value* for probability m. In our measurements, we use $m = 0.9$, 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.

As previously mentioned, the global optimization objective of our algorithms is to minimize the maximum link utilization in the network, i.e. our schedules minimize the ratio of available bandwidth we require from network links to achieve the transfers. Thinking of the CCDFs of available bandwidth distributions in the context of cross-traffic, a desirable effect of our approach is that we maximize the probability that the required value is actually available, thus we maximize the probability that the deadlines are met.

When building the distributions, there is the complementary problem of tuning the measurement framework such that the historical data we use to infer expected values is representative for current values. In the context of the previously observed self-similar nature of cross traffic over several timescales (e.g. typically periodic pattern observed over day-night timeframes) and of other practical factors, such as native layer path flopping and the need to keep the sampling traffic reasonably low, this is a non-trivial problem, that has also been extensively addressed by previous research literature, but is not the focus of the present paper.

5 EVALUATION

We evaluate our system by both trace-driven simulations of the behavior of the scheduling algorithms and real-world evaluation on Planetlab and the Intrigger wide area cluster.

5.1 Simulations

We do trace-driven simulations based on real available bandwidth traces collected on Planetlab and the Intrigger wide area grid. We leverage public Planetlab measurement traces we downloaded from the S3 measurement project, spaning about 7 months worth of measurements. From the S3 traces, we select pathchirp available bandwidth measurements, since we found them to be the most relevant. From these traces, we first derive (almost) full mesh networks of various sizes. We have currently generated networks of 20,40,60 and 80 nodes. Such a network contains N randomly picked nodes, as well as all the links available in the traces that connect these nodes. In principle, they should form a full mesh, but because of node non-availability, problems with pathchirp or perhaps because of the setup of the S3 inference engine, some links are missing in the S3 traces.

Also, for some evaluation setups, we have used overlays that are increasingly well connected rather than full meshes, to estimate the impact of the number of alternate paths in the network on our scheduling algorithms. To obtain increasingly well connected networks, we start from a full mesh and we generate Erdos-Renyi G(n, p) random networks [**?**], with variable values of the connection probability p. E-R graphs are proved to be connected with high probability if the per-node connection probability p is chosen above a threshold of (ln(N)/N). To ensure minimal connectivity, they should thus have a number of links of about N*ln(N), where N is the number of nodes. We variate p, and thus obtain networks in which there are increasingly many alternate paths for a any given request, since the distribution of node degrees in E-R networks is uniform.

Besides network connectivity, another parameter we variate is K, the number of alternate paths that the KWP and KRP heuristics look for. We also variate the network size and the number of requests in the network.

For both simulations and measurements, we use 3 main types of evaluation setup (experiments at setup 4 below use the same networks as setup 2):

- Setup 1: For a network size of 40 nodes and 10 requests, we use either a slightly connected network (p=0.32) or a full mesh (p=1), we evaluate all 5 algorithms and 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). As an aside, for a given network and problem, the same exact solution is valid if the problem is scaled up (or down), e.g. for our setting if we consider windows of 1600 minutes and volumes of 1000-5000 MB (insofar as the timeline granularity and smallest data unit are also scaled up to preserve the same precision). The conclusion of this experiment is that BDRT does much better against the others on the full mesh as compared to the barely connected network, which is intuitive. Also, the heuristic algorithms do better, but are also slower as K increases, which is again intuitive. Results are detailed in subsection 5.1.1.

- Setup 2: For a network size of 40 nodes and 10 requests, we variate the network 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. We evaluate all 5 algorithms. For the heuristic ones, we use a few values of parameter K (5, 30 and 70). Request generation is as before. We can see from the results that, as connectivity increases, BDRT does better, whereas the others do worse, especially for small Ks. Also, all algorithms get

slower as connectivity increases, especially BDRT and BDTS/K*, for big values of K. Results are detailed in subsection 5.1.2.

- Setup 3: We variate the network size and the number of requests. We try networks of 20,40,60 and 80 nodes and numbers of requests of 5, 10 and 10, in all combinations. Let us note that the BDRT equation system is linear in the number of edges in the network and quadratic in the number of requests (and BDTS is linear in the number of paths and quadratic in requests), therefore optimizing the problem for many requests takes a long time, at least in our current, sequential, simplex-based implementation. Results are shown in subsection 5.1.3.

- Setup 4: For the same setting as in Setup 2, we measure some statistics that give us insight into how the various algorithms work, e.g. How many paths does the optimal solution use on average? How many hops do they have ? How do algorithms use their active window? The list of metrics me measured are are: a) routing-related statistics: number of paths used to stream in parallel at the same time (i.e. by a given request in a given time interval), number of hops paths have, number of path flops = changes of path used from one time interval to another by a given request) and b) time-domain scheduling related statistics = percentage of the legal active window actually used in active sending, number of disjoint non-zero intervals the active window was split into, average bandwidth that the transfer used = volume transfered divided by time spent in active sending, the distributions of times that the start of a request sending was delayed by and the distribution of times that requests were scheduled to finish before their deadline). We chose Setup 2 to derive this statistics in order to see how these metrics are influenced by variable path diversity in the network. Results are shown in subsection 5.1.4.

5.1.1 Variable K parameter

Figures 2 and 3 show the congestion factor achieved by all algorithms in the minimal connectivity network, respectivelly in the full mesh. First, we can note than both oblivious approaches achieve congestions at least an order of magnitude worse than their non-oblivious counterparts. We see that both algorithms improve as K grows. Also, KRP seems to be the better strategy in high connectivity networks, and KWP seems to do better in low connectivity ones. We believe this is due to the fact that, when there are more alternate paths, KWP tends to bias towards the same few ”fat” paths to well-connected relays, thus increasing congestion on them, whereas KRP does not have this problem. On the other hand, when there are not as many alternate paths, KWP wins by choosing wider paths than KRP’s random choice. Figures 4 and 5 show the same graphs but zoomed in by only keeping the more effective non-oblivious algorithms. We can see that the same observation about KWP vs. KRP holds for the time-optimized cases as well. We can also see that both heuristic strategies, although improving as K grows, remain sub-optimal and that the difference between BDRT and them is bigger for the high connectivity network. Especially KWP/BDTS performs bad on the full mesh, as well as KRP/BDTS on the low connectivity network. Except for K=5, KRP/BDTS performs much better, in absolute terms, at high connectivities than at low ones. These results are intuitive for the same reason related to KWP’s bias and KRP’s random choice. BDRT seems to effectively take advantage of the high path diversity in the full mesh. Figures 6 and 7 show the runtimes of the algorithms in the same setup. In our current implementation, we used a sequential linear program solver based on glpk’s standard implementation of the simplex method. We did do some glpk tunning for speed, such as using expensive exact arithmetics only to check final solutions (the -xcheck option), solving the dual equation system (-dual) and choosing appropriate data granularity. The node 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 step and might be interesting as future work, but was not the focus of the current paper. As we can see, runtimes increase as K increases and the full mesh takes longer to solve than the low connectivity network. We can see that BDRT is slower up to the point where K=70 (low connectivity network), respectivelly K=50 (full mesh), at which point BDTS/* become comparable or even slower. BDTS/KRP is slower than BDTS/KWP on the full mesh, whereas the order changes on when connectivity is low for K=70. The heuristic strategies are considerably faster (since the linear system to solve is much simpler than BDRT and BDTS). OBLIVIOUS/KRP is the most efficient in runtime and also scales best in runtime as K grows. OBLIVIOUS/KWP is next faster and also scales well at high connectivity and a bit worse at low connectivity. BDTS/* perform quite close at low connectivity, with a slight edge for BDTS/KWP, especially for higher values of K. Comparing both congestion factor and runtime graphs, we can argue that the heuristic strategies can in principle get close to BDRT for high values of K and aided by BDTS time-domain optimization (although BDRT still provides a factor of 2 improvement in the high connectivity case), but at these high values their runtimes become similar or slower than BDRT as well. The only case where it might make sense to replace BDRT is at low connectivities and replacing it with BDTS/KRP, which achieves similar performance at improved time cost.

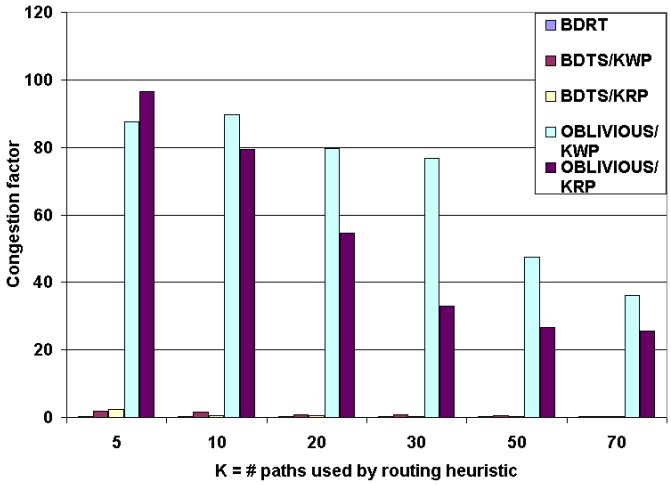

Figure 2: High connectivity, variable K, all algorithms - Congestion factor.

5.1.2 Variable network connectivity

Figures 8, 10, 12 and 14 plot the congestion factor achieved by BDTS/KWP, BDTS/KRP, OBLIVIOUS/KWP and OBLIVIOUS/KRP against BDRT, respectively, when varying the net-

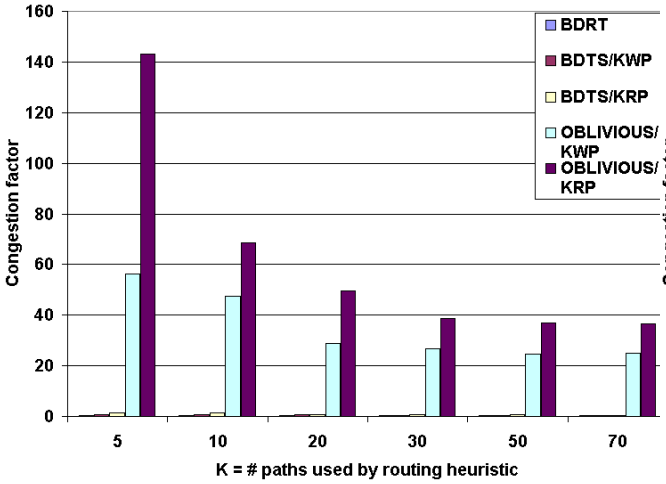

Figure 3: Low connectivity, variable K, all algorithms - Congestion factor.

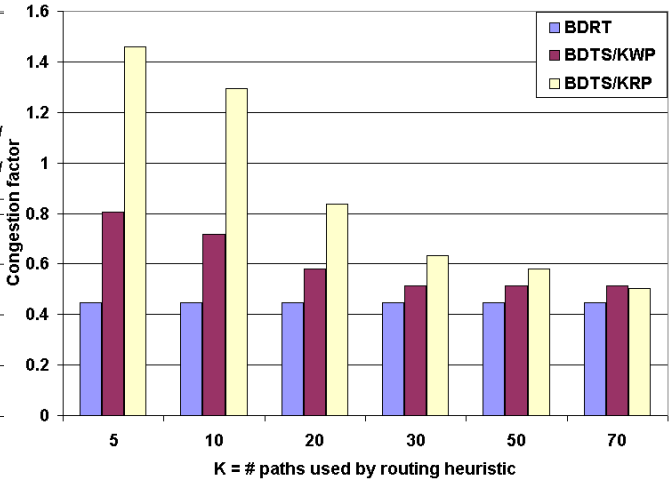

Figure 5: Low connectivity, variable K, non-oblivious algorithms - Congestion factor.

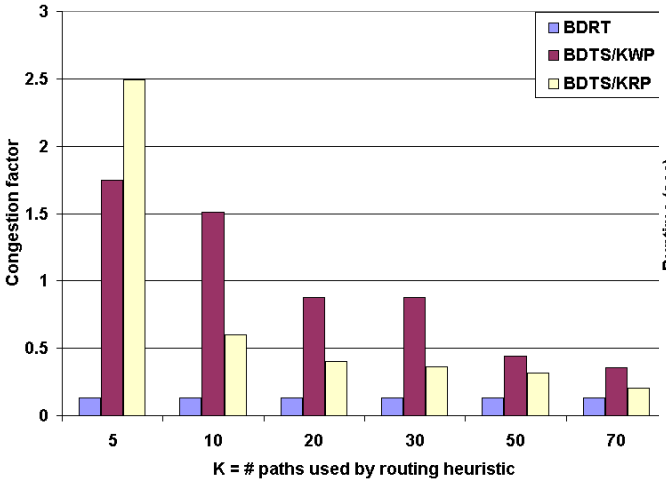

Figure 4: High connectivity, variable K, non-oblivious algorithms - Congestion factor.

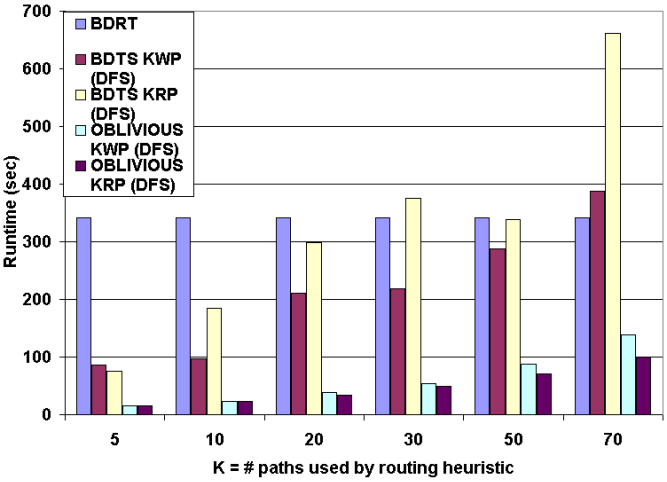

Figure 6: High connectivity, variable K, all algorithms - Runtime.

work connectivity from minimal to full mesh, by varying the E-R random network connection probability p. As in all experiments, all algorithms are given identical sets of problems to solve. In each graph, there are 3 columns for the heuristic algorithm, one for each K=5,30 and 70. In all graphs, we can see clearly that BDRT improves as the network has more alternate paths. The heuristic algorithms generally worsen their performance as the network gets more connected. This is always the case with the 2 oblivious algorithms and also for the BDTS-aided ones, for K=5 and K=30. For K=70, BDTS/KRP levels its performance, whereas BDTS/KWP initially improves, but then worsens again at very high connectivities. The */KRP algorithms scale better than */KWP wrt connectivity. Whereas the KWP-based ones have a clearly linear loss of performance, with the slope getting better as K increases, the KRP-based ones have an inherent variability that makes the dependence less clear. This variability is probably inherent to the random choice of paths that the algorithm makes. On the other hand KWP's loss of performance is probably due to the same fact that wide paths get overbooked as the connectivity increases. Again we see that all algorithms are sub-optimal, the oblivious algorithms fare much worse than the non-oblivious. Figures 9, 11, 13 and 15 show the runtimes for this setting. We can see that runtime increases as there are more links, that

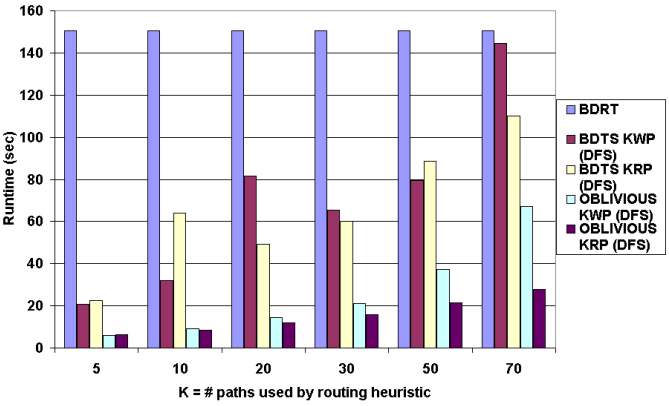

Figure 7: Low connectivity, variable K, all algorithms - Runtime.

there is some variability especially for high K (i.e. as the linear program analyzes more paths), probably due to the simplex algorithm we use in the current implementation.

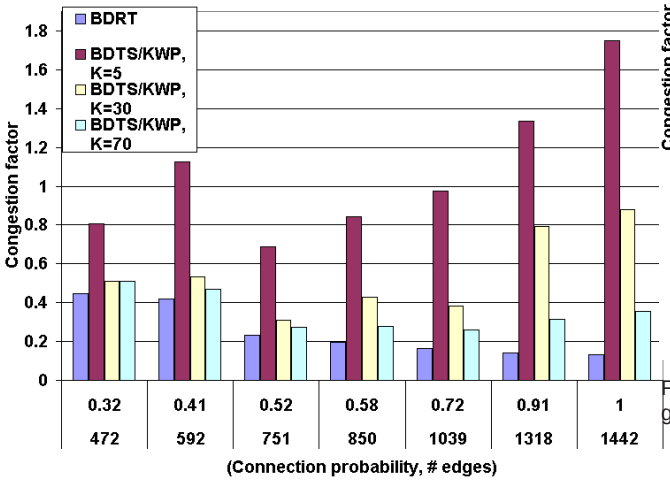

Figure 8: Variable connectivity, BDTS/KWP, K=5,30,70 - Congestion factor.

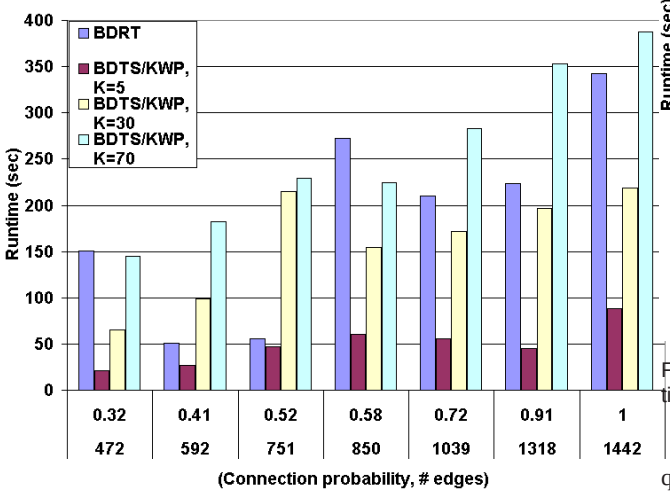

Figure 9: Variable connectivity, BDTS/KWP, K=5,30,70 - Runtime.

5.1.3 Network size and number of requests

We will present the measurements for this setup in the final version of the paper, to be submitted to ICCCN 2009, in early March.

5.1.4 Insights on routing and time domain scheduling

Figures 16 and 17 show CDFs of the number of paths used to stream a request in parallel at the same time by BDRT and KWP/KWP (K=5) algorithms. We can see that BDRT tends to take advantage of the extra connectivity by streaming on more paths in parallel as the path diversity increases. The BDRT CDF shows clearly how the bigger the connectivity is, the higher the number of paths tends to be. On the contrary, the BDTS/KWP algorithm tends to use less paths as connectivity grows. We believe this is due to the fact that the

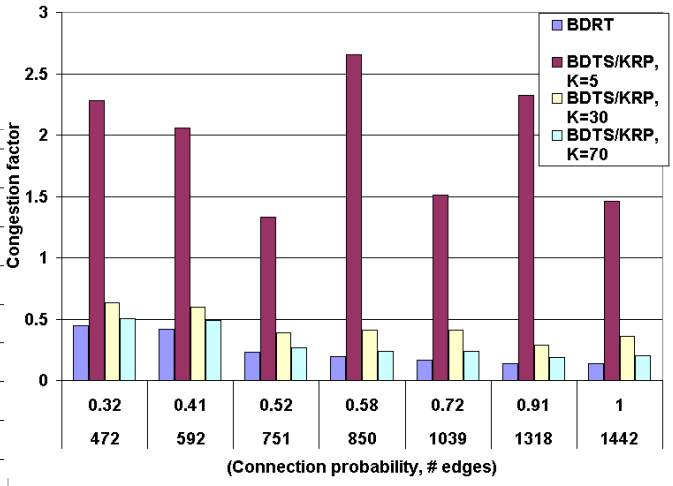

Figure 10: Variable connectivity, BDTS/KRP, K=5,30,70 - Congestion factor.

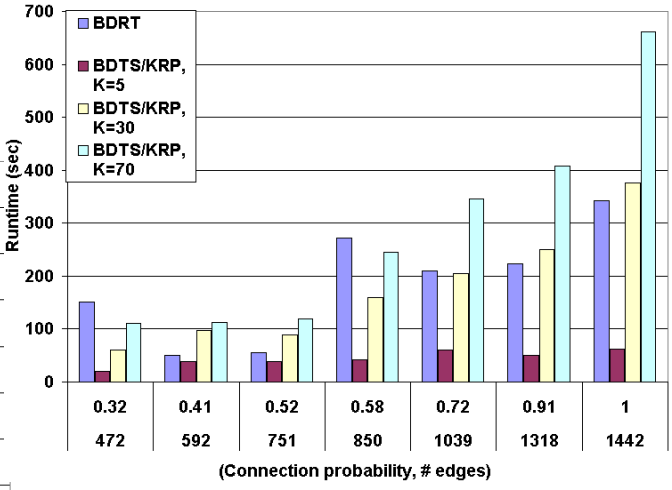

Figure 11: Variable connectivity, BDTS/KRP, K=5,30,70 - Runtime.

quality (=bandwidth) of paths returned by KWP increases as the connectivity increases, therefore BDTS only needs to use few paths to achieve the optimum congestion achievable given those few paths. In contrast, as connectivity increases, BDRT does better, since it can take advantage of the extra bandwidth by actually taking into account all the paths in the network, without being biased towards some paths or links. BDTS/5WP uses an average of 2-2.5 paths in parallel, whereas the optimal solutions output by BDRT use more paths, with averages ranging from 24 (p=0.32) to 204 (p=1, full mesh), as can be seen from Fig. 18, showing the average values. However, as it can be seen from the CDF graphs, especially the BDRT distribution is heavy-tailed, i.e. those averages are biased by a few high values in some instances, while the rest have smaller values. Specifically, the minimum number of paths used by BDRT in all cases is 1 and the maximum ranges from 226 (p=0.32) to 1391 (p=1). The distribution seems to be more heavy tailed for small connectivities, for BDRT. On the other hand, the BDTS CDF shows that BDTS is less sensitive in this respect to connectivity, and it only tends to use slightly less paths for either high connectivities (0.91 and 1) or low ones

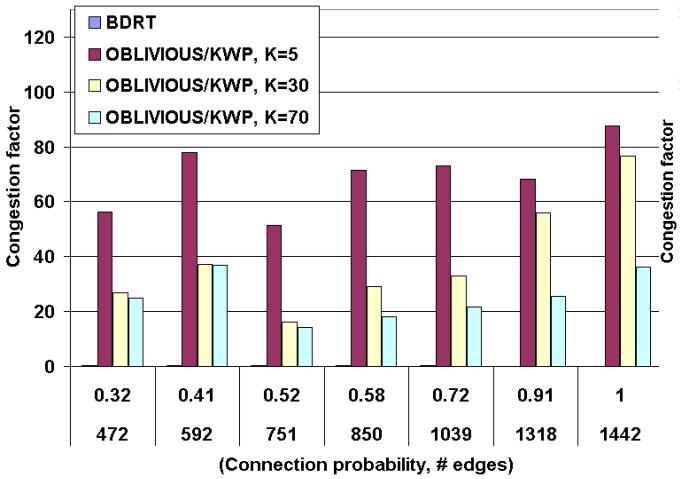

Figure 12: Variable connectivity, OBLIVIOUS/KWP, K=5,30,70 - Congestion factor.

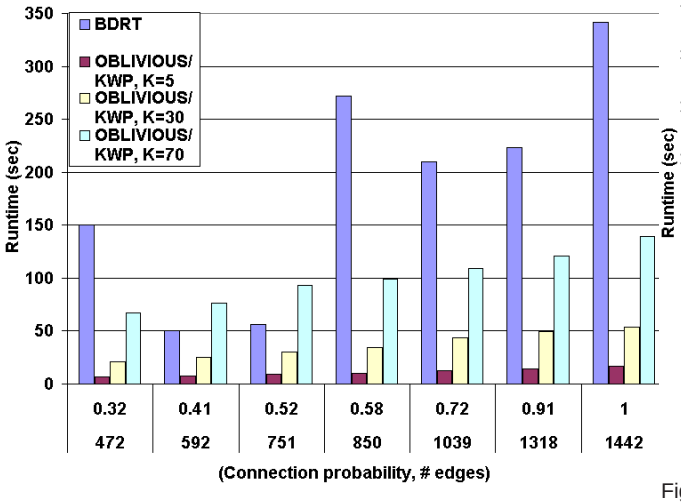

Figure 13: Variable connectivity, OBLIVIOUS/KWP, K=5,30,70 - Runtime.

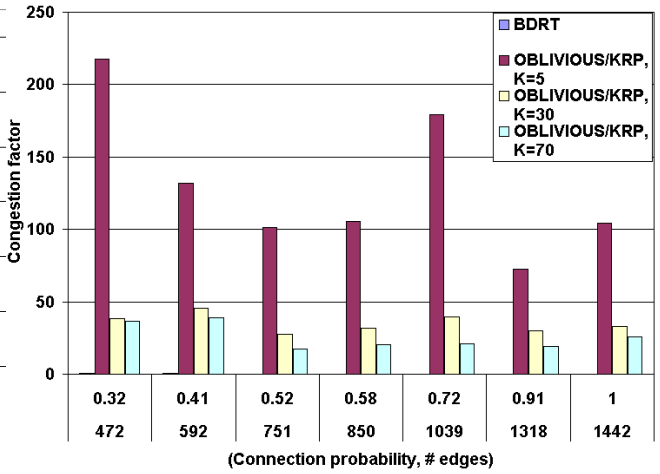

Figure 14: Variable connectivity, OBLIVIOUS/KRP, K=5,30,70 - Congestion factor.

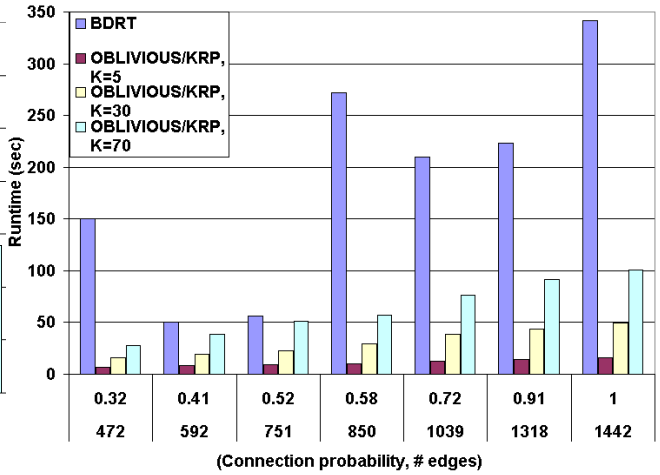

Figure 15: Variable connectivity, OBLIVIOUS/KRP, K=5,30,70 - Runtime.

(0.32 and 0.41) Figure 23 shows the average number of hops that paths have, for each connectivity value, for BDRT and BDTS/5WP. We can see is that the number of hops seems not to be influenced by connectivity, at least in the case of BDRT. This is confirmed by the CDFs (not included due to space limitations). The influence on BDTS/KWP seems to exist, with more connectivity incurring shorter path lengths. This is not so clear from the average, but it visible in the CDF (not included). Indeed, the more connectivity, the less hops BDTS/KWP seems to use. This is probably because, as connectivity grows, it is more likely that KWP finds more direct paths to the most well connected nodes, thus shorter widest paths.

Figures 19, 20 and 21 show us that BDRT does much more path flops (i.e. path changes to send the same request in different intervals) - averages from 55 to 601 - than BDTS/KWP - averages 1.4 to 3.1, and that BDRT is strongly influenced by connectivity (higher connectivity implies more path flops), whereas BDTS/KWP is not (very slightly less flops for high connectivities). These conclusions are confirmed by both average values and CDFs (average values graph for BDTS/KWP ommited due to space limitations).

For the time domain metrics, in the case of average percentage of active window used, measurements showed us that BDRT seems to be unsensitive to connectivity in this respect (keeping averages between 72% and 76%), BDTS/KWP seems to also be uninfluenced by connectivity, except at high values of p (p=0.91 and p=1). BDTS/KWP uses slightly more of the active window than BDRT, between 75% and 78%). This probably happens because the paths returned by KWP at high connectivities are wide enough that for BDTS to reach optimal congestion given those paths, therefore it is enough to send only in a smaller portion of the active window. For BDRT the same doesn't happen, since it doesn't only use the widest paths, even if they are available. The CDFs for these distributions are not shown, but all they do besides confirming these conclusions is to show that the distributions are not heavy-tailed, but rather uniform.

Regarding the number of disjoint sending intervals (i.e. disjoint time intervals in the active window when there is at least a path in the network for which the transmission rate for that request is > 0.0) that the 2 algorithms used at various connectivities, we noticed that BDRT is not influenced by connectivity and that it tends to use slightly more intervals than

BDTS/KWP (1.4 to 1.9 versus 1.2 to 1.7). BDTS/KWP seems to use more intervals for highly connected networks, probably due to the fact that the conflict between requests increases there, due to the fact that requests have to share the same widest links. Therefore, BDTS has more "work" to do.

The "average bandwidth used" metric is related (negatively correlated) to the percentage p of active window used $(= \nu_r/(p\omega_r))$. From Figures 24 and 25, we can see that BDRT is insensitive, BDTS only uses slightly more bandwidth at higher connectivities (with the same explanation as in the case of percentage of active window used). Also, the 2 algs are quite similar in this respect, with only slightly more bwidth used by BDRT at very high, as can be seen from Figure 26, showing the average values. It is interesting that BDRT manages to do this while still achieving lower congestion.

Regarding the average time by which starts were delayed and by which deadlines were scheduled to finish earlier our conclusion is that these metrics seem to be uncorrelated with network connectivity, especially BDRT. BDTS/KWP tended to delay transfers and finish them earlier slightly more for high connectivities, probably reflecting the fact that more time-domain optimization needs to be done there, as explained.

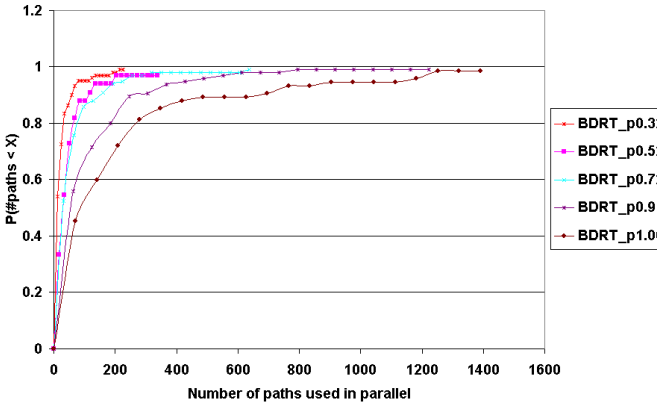

Figure 16: Variable connectivity, CDFs of number of paths used in parallel for a single request, BDRT.

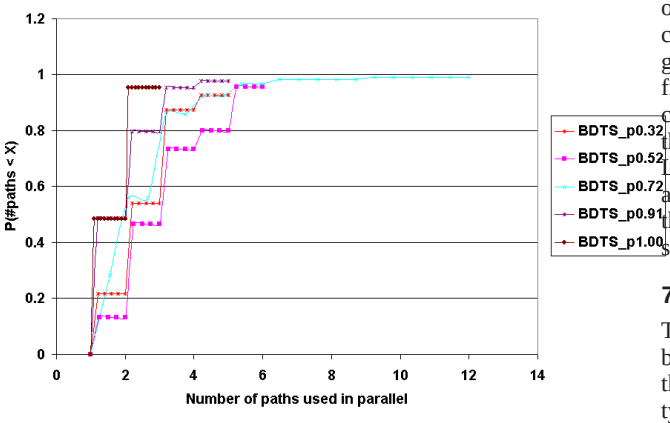

Figure 17: Variable connectivity, CDFs of number of paths used in parallel for a single request, BDTS/5WP.

5.2 Real-world transfers

We will present the transfer measurements results on PlanetLab and Intrigger in the final version of the paper, to be submitted to ICCCN 2009, in early March.

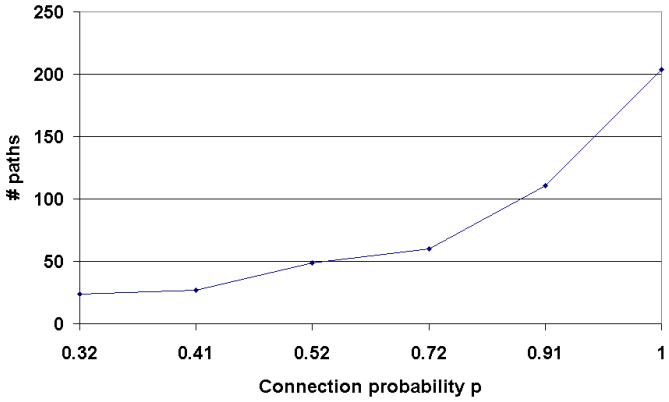

Figure 18: Variable connectivity, average number of paths used in parallel for a single request, BDRT.

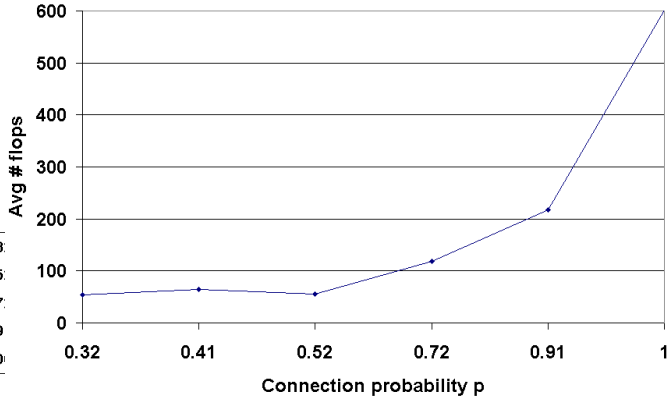

Figure 19: Variable connectivity, average number of path flops during transfer, BDRT.

6 Related work

Racke has found [16] that, for generic graphs, there exist oblivious routing algorithms that achieve a polylogarithmic competitive ratio w.r.t. congestion, that is, can achieve congestion that is within a polylogarithmic factor ($O(\log^3(n))$) from optimal. He bases his proof on producing a tree decomposition of the network, and then showing that routing on the tree is almost as good as routing in the original network. Later Bienkowski et al [6] and, independently, Harrelson et al. [11] have proposed polynomial-time oblivious algorithms that are $O(\log^4(n))$, and $O(\log^2(n)\log(\log(n)))$-competitive, respectively.

7 Future work and possible applications

The outcome of our implementation is a global scheduler for bulk data transfers, to be used by applications using overlays that do application-level routing over public networks. The type of applications targeted are mainly those mentioned in the introduction section, primarily including novel cloud computing or streaming applications in which peers need to cooperatively reach a common goal, such as jointly transfering and processing sets of distributed data.

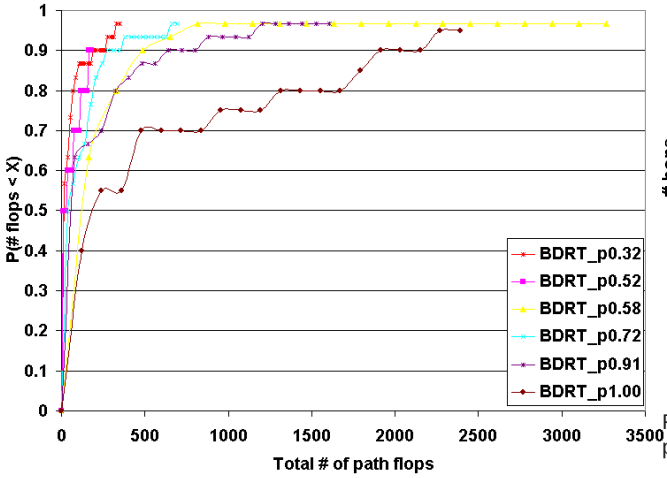

Figure 20: Variable connectivity, CDFs of number of path flops during transfer, BDRT.

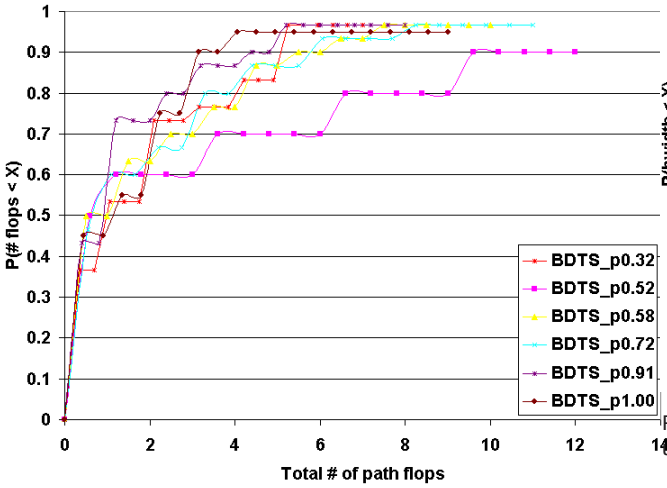

Figure 21: Variable connectivity, CDFs of number of path flops during transfer, BDTS/5WP.

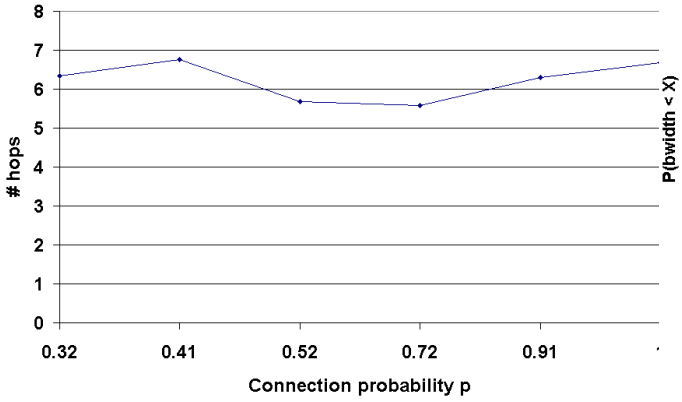

Figure 22: Variable connectivity, average number of hops that paths have, BDRT.

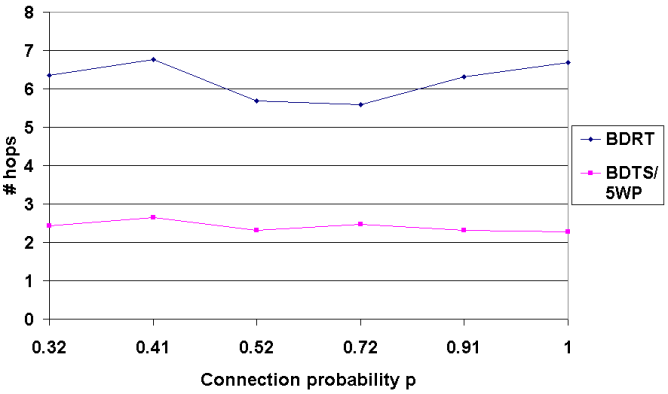

Figure 23: Variable connectivity, average number of hops that paths have, BDRT and BDTS/5WP.

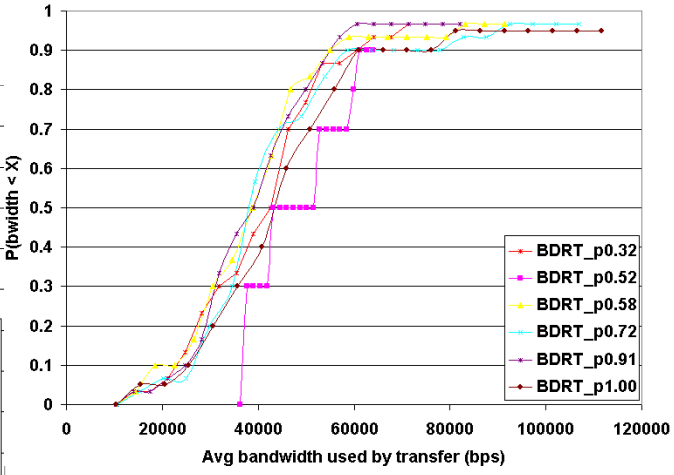

Figure 24: Variable connectivity, CDFs of average bandwidth used, BDRT.

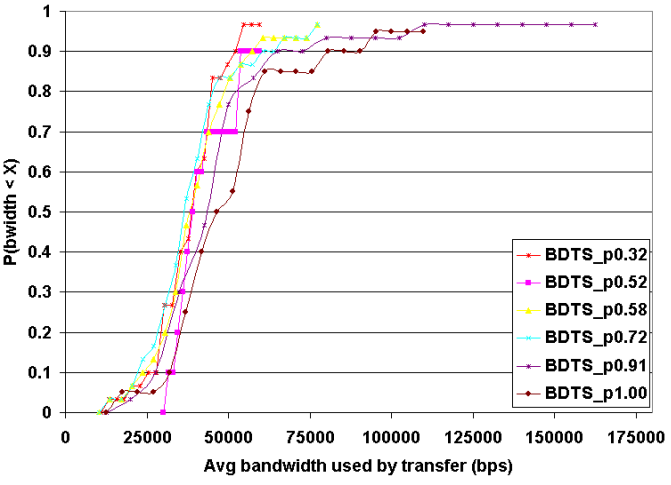

Figure 25: Variable connectivity, CDFs of average bandwidth used, BDTS/5WP.

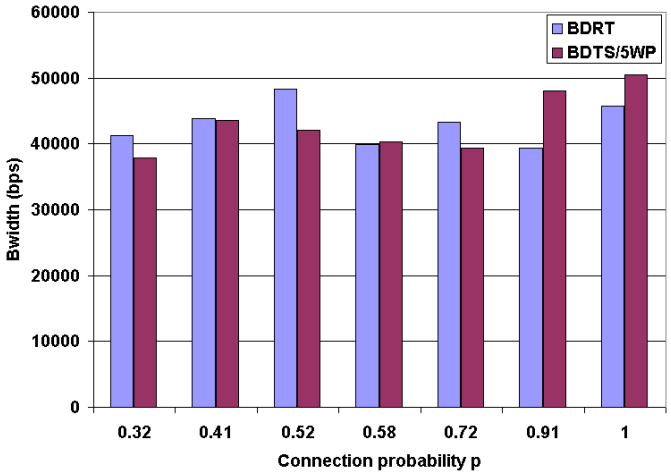

Figure 26: Variable connectivity, average bandwidth used, BDRT and BDTS/5WP.

REFERENCES

[1] Bittorrent, www.bittorrent.com.

[2] Lofar, www.lofar.org.

[3] Seti@home, http://setiathome.berkeley.edu.

[4] Ska, www.skatelescope.org.

[5] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of SOSP*, October 2001.

[6] M. Bienkowski, M. Korzeniowski, and H. Racke. A practical algorithm for constructing oblivious routing schemes. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[7] Binbin Chen and Pascale Vicat-Blanc Primet. Scheduling bulk data transfers in grid networks. 2007. IEEE CCGRID2007.

[8] F. Douglis, M. Branson, K. Hildrum, B. Rong, and F. Ye. Multi-site cooperative data stream analysis. In *SIGOPS Oper. Syst. Rev.*, 2006.

[9] R. Guerin and A. Orda. Networks with advance reservations: The routing perspective. In *Proceedings of IEEE INFOCOM*, 2000.

[10] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: efficient algorithms and complexity.

[11] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[12] P. Karbhari, M. Ammar, and E. Zegura. Optimizing end-to-end throughput for data transfers on an overlay-tcp path. In *Proceedings of IFIP Networking Conference*, 2005.

[13] Sung-Ju Lee, Sujata Banerjee, Puneet Sharma, Praveen Yalagandula, and Sujoy Basu. Bandwidth-aware routing in overlay networks. In *Proceedings of INFOCOM*, 2008.

[14] C. Martel. Preemptive scheduling with release times, deadlines, and due times. In *Journal of ACM*, volume 29, pages 812–829, July 1982.

[15] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips. Tribler: A social-based peer-to-peer system. In *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[16] H. Racke. Minimizing congestion in general networks. In *Proceedings of FOCS 43*, 2002.

[17] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *In Passive and Active Measurement Workshop (2003).*, 2003.

[18] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. Overqos: An overlay based architecture for enhancing internet qos. In *Proceedings of NSDI*, 2004.

[19] Y. Wang and Z. Wang. Explicit routing algorithms for internet traffic engineering. In *Proceedings of ICCCN*, September 1999.

[20] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee. S3: a scalable sensing service for monitoring large networked systems. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, 2006.