Scheduling and transport for file transfers on high-speed optical circuits

M. Veeraraghavan & Xuan Zheng, Univ. of Virginia, {mv5g,xuan}@virginia.edu
Wu-chun Feng, Los Alamos Natl. Lab, feng@lanl.gov
Hojun Lee, Polytechnic Univ, hlee@photon.poly.edu
Edwin Chong & Hua Li, Colorado State Univ., {echong,huali}@engr.colostate.edu

Abstract[1]: Scheduling resources on Grids is a well-known problem. The extension of Grids to LambdaGrids requires the scheduling of lambdas, i.e., end-to-end high-speed circuits. In this paper, we propose a scheduling heuristic for such lambdas in support of large-scale scientific applications that require high-throughput transfers of large files. We refer to this heuristic as "Varying-Bandwidth List Scheduling" (VBLS) because the scheduler returns a Time-Range-Capacity (TRC) allocation vector with varying bandwidth levels assigned for different time ranges within the duration of a transfer. The advantage of VBLS over a fixed-bandwidth allocation scheme is that it allows the scheduler to backfill any holes left in resource allocations. Enabling VBLS requires end host applications to specify the file size in their transfer requests. To characterize VBLS, we ran simulation experiments that show that VBLS performance approaches packet-switching performance. This result means that file transfers can take advantage of bandwidth that becomes available subsequent to the start of transfers, a current and critical drawback of typical fixed-bandwidth allocation schemes in circuit-switched networks. Next, we identify the key features needed in a transport protocol that works in conjunction with VBLS and develop the "Varying Bandwidth Transport Protocol" (VBTP). VBTP is a rate based flow control scheme that is coupled with Selective-ARQ based error control. Finally, the paper concludes with a discussion on the impact of transport problems on VBLS scheduling.

Keywords: Scheduling, Simulation, Transport protocols, File transfers, Optical networks, Circuit switching

1. Introduction

A need for high-speed networks that can offer deterministic bandwidth and latency has been identified for e-Science projects that involve geographically distributed scientists [1]. A number of optical network testbeds, such as Canarie's CA*net 4, Starlight, UKlight and SURFnet, have been deployed to meet this need [2]. These networks allow for end-to-end circuits to be provisioned for the dedicated use of an application. Some of these networks use all-optical switches with the granularity of

1. This work was carried out under the sponsorship of NSF grants, ITR-0312376 and AIN-0335190 at UVA, grant 0087487 at Polytechnic University, grants ECS-0098089, ANI-0099137, and ANI-0207892 at Colorado State University, and was supported partly by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36.

a circuit being a single wavelength, while others use hybrid electronic/optical switches that provide sub-lambda granularity.

The Grid community, a community that has long addressed the computing and storage needs of large-scale scientific applications, has coined the term "LambdaGrid" to characterize a grid in which "lambda networks" interconnect geographically distributed computing and storage resources. The term "lambda" is used to describe an end-to-end circuit, which typically is a concatenation of 1Gbps Ethernet (GbE) or 10Gbps Ethernet (10GbE) signals from/to end hosts mapped onto wide-area SONET/SDH circuits or all-optical lightpaths, depending upon whether the switches are electronic, time-division multiplexed (TDM), SONET/SDH switches or all-optical, wavelength-division multiplexed (WDM) switches.

A number of problems need to be solved before scientists can start enjoying the connectivity offered by these networks in a seamless fashion. These include (i) dynamic provisioning of these end-to-end circuits, (ii) Authentication, Authorization, Accounting (AAA) for user requests, (iii) transport protocols on these end-to-end circuits for different types of applications, and (iv) algorithms to schedule circuits ([1] notes that "lambdas" are resources that can be scheduled like any other computing or storage resource). Significant progress has been made on some of these problems.

In this paper, we focus on the **lambda scheduling problem** and the **transport protocol** problem. We further limit our problem statement to handling only file transfers. The file-transfer application is clearly of interest to geographically-distributed scientists who often need to download large-sized (e.g., terabyte-sized) files from remote sites [3]. The other class of applications of interest to scientists is distance (or remote) visualization, computational steering, and collaboration. We relegate extending our scheduling algorithm to include requests for lambdas for such applications and the development of a corresponding transport protocol to a future paper.

We formulate our **scheduling** problem as follows. End host applications request lambdas for file transfers by specifying a three-tuple: $(F^i, R^i_{max}, T^i_{req})$, where F^i is the file size of the ith request, R^i_{max} is a maximum bandwidth limit for this request, and T^i_{req} is the desired start time for the transfer. In theory, file transfers can be carried out at any data rate. However, in practice, various constraints of the end hosts, such as disk speeds, bus rates, processing speeds and memory access rates, limit the maximum rate possible. Hence, we require users to specify this maximum rate and then have the scheduler assign bandwidth at some rate lower than or equal to R^i_{max}.

Requiring applications to specify file sizes is atypical of today's file-transfer implementations such as FTP or GridFTP. However, this information is readily available at file servers. Given that applications such as FTP or GridFTP already need to

be upgraded anyway to work with transport protocols designed for lambdas, having file servers make scheduling requests on behalf of their clients can be added as part of these upgrades. The benefit is that it allows our scheduler to provide a varying-bandwidth allocation instead of a single fixed-rate allocation for the duration of the transfer. We call this varying-bandwidth allocation a Time-Range-Capacity (TRC) vector. A TRC vector $\boldsymbol{TRC}^i$ allocated for the i th transfer is characterized as follows: $\{(B_k^i, E_k^i, C_k^i), k = 1, \ldots, \tau^i\}$, where B_k^i is the start of the k th time range, E_k^i is the end of the k th time range and C_k^i is the capacity allocated for the transfer in the k th time range. The reason for making such a TRC allocation is that it allows the scheduler to backfill any holes left in resource allocations from other requests. In doing so, we overcome the often-quoted disadvantage of circuit-switched networks that once a file transfer is assigned a certain bandwidth it cannot take advantage of bandwidth that becomes available subsequent to its start in contrast to packet-switched networks [4]. The scheduler is able to make a TRC allocation for an incoming request because it knows the file sizes and TRC allocations of all ongoing transfers. We call this algorithm **Varying Bandwidth List Scheduling (VBLS)**. The cost of implementing VBLS is that the circuit switch needs to be reprogrammed multiple times within a transfer unlike in the fixed-bandwidth allocation mode where the switch is only programmed at the start and end of a call. Requiring users to provide file sizes could be compared to requiring computer users to specify job lengths to computing resource schedulers. However, an important difference is that while job lengths are hard to predict [5], file sizes are clearly known.

We formulate our **transport protocol problem** as follows. Since hosts that can be interconnected via an end-to-end high-speed circuit on one of these LambdaGrids can be expected to also have connectivity via the Internet, we design a transport solution that uses a combination of these two paths. The high-speed dedicated circuit is held open only for as long as user data flows from the server to the client. It is used only for the actual data transfer and for retransmissions that are identified as being required prior to the completion of the file transfer. The circuit is not held open after completion of the transfer to verify that the final few blocks sent on the circuit have indeed reached the client successfully because this would lead to the circuit lying idle. The two functions needed in a transport protocol for this combination of a dedicated circuit and TCP/IP path are error control and flow control. The error control scheme is a selective-ARQ scheme, while the flow control scheme is rate based. The transport protocol module at the sending end-host uses the TRC allocation assigned by the VBLS scheduler for the circuit to dynamically adjust its sending rate in the rate-based, flow-control scheme. We call this transport protocol "**Varying Bandwidth Transport Protocol" (VBTP).** Since congestion is handled during circuit setup, once the circuit is successfully provisioned, congestion control functionality is not required during the data transfer.

The rest of the paper is organized as follows. Section 2 describes related work on the topics of scheduling and transport protocols for file transfers on circuits. Details of the VBLS scheme are then presented in Section 3 along with simulation results that demonstrate the benefits of using a varying-bandwidth allocation scheme over a fixed-bandwidth allocation scheme. Section 4 describes our transport protocol. We conclude the paper in Section 5.

2. Related work

There is clearly a rich heritage on scheduling in many contexts, including job-shop scheduling [6], scheduling computing resources in general and in the Grid setting in particular (e.g., Condor [5] and Globus [7] projects, respectively), and scheduling file transfers [8-10]. File-transfer scheduling algorithms are primarily **List Scheduling (LS)** schemes that work as follows: if there is a call i such that its required bandwidth b_i is available on all links of the end-to-end path, LS schedules the first such call in the list of all calls. If not, it waits until an active transfer completes. That is, LS is a greedy approach to scheduling. Our scheduling solution builds on this basic LS scheme. Scheduling intrinsically involves booking ahead, i.e., making advance reservations. Papers on this topic [11-13] are also considered in developing our algorithm.

A number of new transport protocols have been proposed for high-speed networks to run on top of UDP and have been implemented as application-level processes. Examples include SABUL [14], UDT [15], Tsunami [16], and RBUDP [17]. Others have enhanced TCP [18-20] and implemented these enhancements in the kernel space. Most of these enhancements are for high-speed packet-switched networks, which means they run congestion-control mechanisms to adjust sending rates during data transfers based on congestion levels. RBUDP is an exception, being specifically targeted at photonic networks. Another class of transport protocols, such as Scheduled Transfer (ST) [21] and RDDP [22], has been designed for OS-bypass implementations. At the 10Gbps rates, we expect bottlenecks within end hosts to require OS-bypass implementations. We use concepts from all of these protocols in developing our transport solution called the Varying Bandwidth Transport Protocol (VBTP).

VBTP works in conjunction with VBLS to make rate adjustments in accordance with the TRC allocation for the circuit. Many UDP-based transport protocols, such as SABUL, Tsunami, and RBUDP, support rate control. We present results of our experiments with these protocol implementations to verify whether end hosts can indeed maintain the negotiated rates in spite of the variability induced by other processes.

We contrast this approach of using VBLS and VBTP with some of the ongoing efforts to enhance TCP performance such as

FAST [20]. Because the IP network is connectionless, which means it does not allow for a priori reservations of bandwidth, FAST and TCP Vegas [23] are based on taking constant measurements of available bandwidth with corresponding adjustments of the sending rate. These schemes are highly dynamic, and hence, allow a sender to enjoy the advantage offered by packet-switched networks, i.e., the ability of a transfer to take advantage of bandwidth that becomes available after it starts. VBLS/ VBTP achieves the same goal but without the overhead of having to take constant measurements by engaging in a priori reservations.

3. VBLS: A Lambda-Scheduling Algorithm for File Transfers

In this section, we describe our VBLS scheme for scheduling calls on an m-channel single link. Table I lists our notation.

TABLE I NOTATION

Symbol	Meaning
F^i	File transfer size requested by call i
T^i_{req}	Start time requested for call i
R^i_{max}	Maximum rate requested for call i expressed as a number of channels; typically limited by access link rate or end host processing rates
$\boldsymbol{TRC}^i = \{(B^i_k, E^i_k, C^i_k), k = 1, \ldots, \tau^i\}$	Time-Range-Capacity allocation: Capacity C^i_k is assigned to call i in time range k starting at B^i_k and ending at E^i_k.
$\gamma(t)$	Capacity availability function: Total number of available channels at time t.
$\gamma(t)$ is expressed in the following form: $\begin{cases} m_z & P_z \le t < P_{z+1} \\ m & t \ge P_{z_{max}} \end{cases}$ where $m_z \le m$ and $z = 1, 2, \ldots, z_{max}$; see Figure 1 for an example.	z_{max} denotes the number of times $\gamma(t)$ changes value before reaching m at $t = P_{z_{max}}$ after which all m channels of the link remain available
χ	Per-channel bandwidth

3.1 VBLS overview

The switch into the LambdaGrid maintains a total available-bandwidth function $\gamma(t)$. Given it knows the **TRC** allocations for all scheduled lightpaths, it knows when and how much link bandwidth is available for new requests. A request i specifies $(F^i, R^i_{max}, T^i_{req})$. The switch's response is $\boldsymbol{TRC}^i$, which is an allocation of capacity for different time ranges for request i.

Bandwidth allocation for a new request is made on a round-by-round basis, where a round consists of the procedures used to allocate capacity for a time range that extends between two consecutive change points in $\gamma(t)$. In a time range between two consecutive change points, we determine whether the entire remaining file can be transferred or the holding time ends within this time range, and whether the available capacity is greater than or less than/equal to R_{max}^i. We define four cases corresponding to the four possible outcomes of these two decisions. At the end of each round, we compute the remaining size of the file or remaining holding time and start the next round.

3.2 Detailed description of VBLS

Start algorithm: Set time $\upsilon \leftarrow T_{req}^i$ *and remaining file size* $\phi \leftarrow F_{req}^i$*;* $k \leftarrow 1$.

Repeat loop (start next round):

Find z such that $P_z \le \upsilon < P_{z+1}$ in the capacity availability function $\gamma(t)$. If $\gamma(\upsilon) = 0$, then reset $\upsilon \leftarrow P_{z+1}$. Continue repeat loop (start next round).

Case 1: Number of available channels is less than/equal to R_{max}^i, and the whole file can be transmitted before the next change in the total available bandwidth curve, i.e., $\gamma(\upsilon) \le R_{max}^i$ and $(P_{z+1} - \upsilon)\gamma(\upsilon)\chi \ge \phi$, then

- Set $B_k^i \leftarrow \upsilon$, $E_k^i \leftarrow \upsilon + \phi/(\gamma(\upsilon)\chi)$, $C_k^i \leftarrow \gamma(\upsilon)$ (the begin time, end time and capacity allocation for the k th range of file transfer i). Set $\tau^i \leftarrow k$ (total number of time ranges allocated to file transfer i). Terminate repeat loop.

Case 2: Number of available channels is less than/equal to R_{max}^i, but the whole file cannot be transmitted before the next change in the total available bandwidth curve, i.e., $\gamma(\upsilon) \le R_{max}^i$, and $(P_{z+1} - \upsilon)\gamma(\upsilon)\chi < \phi$, then

- Set $B_k^i \leftarrow \upsilon$; $E_k^i \leftarrow P_{z+1}$, $C_k^i \leftarrow \gamma(\upsilon)$,
- Set $k \leftarrow k + 1$, $\upsilon \leftarrow P_{z+1}$, and $\phi \leftarrow \phi - (P_{z+1} - \upsilon)\gamma(\upsilon)\chi$. Continue repeat loop (start next round).

Case 3: Number of available channels is greater than R_{max}^i, and the whole file can be transmitted before the next change in the total available bandwidth curve, i.e., $\gamma(\upsilon) > R_{max}^i$, and $(P_{z+1} - \upsilon)R_{max}^i\chi \ge \phi$, then

- Set $B_k^i \leftarrow \upsilon$, $E_k^i \leftarrow \upsilon + \phi/(R_{max}^i\chi)$, $C_k^i \leftarrow R_{max}^i$. Set $\tau^i \leftarrow k$. Terminate repeat loop.

Case 4: Number of available channels is greater than R_{max}^i, and the whole file cannot be transmitted before the next change in the total available bandwidth curve, i.e., $\gamma(\upsilon) > R_{max}^i$, and $(P_{z+1} - \upsilon)R_{max}^i\chi < \phi$, then

- Set $B_k^i \leftarrow \upsilon$, $E_k^i \leftarrow P_{z+1}$, $C_k^i \leftarrow R_{max}^i$. Set $k \leftarrow k + 1$, $\upsilon \leftarrow P_{z+1}$, and $\phi \leftarrow \phi - (P_{z+1} - \upsilon)\chi R_{max}^i$. Continue repeat loop (start next round).

End repeat loop.

As an **example of VBLS**, consider scheduling a transfer of a 5GB file with a T^i_{req} of 50, and an R^i_{max} of 2. Let χ, the per-channel link bandwidth, be 10Gbps and each unit of time correspond to 100ms. Assume the 4-channel link state is as shown in Figure 1. In the time range $50 \le t \le 60$, we can schedule 1 channel for the transfer. Within this range, 1.25GB ($= 10Gbps \times 10 \times 100ms$) can be transferred. In the $60 \le t \le 70$ range, we can allocate 2 channels since R^i_{max} is 2. Therefore we can transfer 2.5GB. The remaining 1.25GB can be assigned to 2 channels past $t = 70$. Even though available bandwidth is 3 channels in the $70 \le t \le 80$, we can only assign two channels because of the R^i_{max} limit. Therefore the TRC vector is as follows: {(50, 60, 1), (60, 70, 2), (70, 75, 2)} where each tuple is of the form (B^i_k, E^i_k, C^i_k) and the number of ranges for this call τ^i is 3. Note that the transfer completes in the middle of the $70 \le t \le 80$ range as indicated by the shaded area in Figure 2.

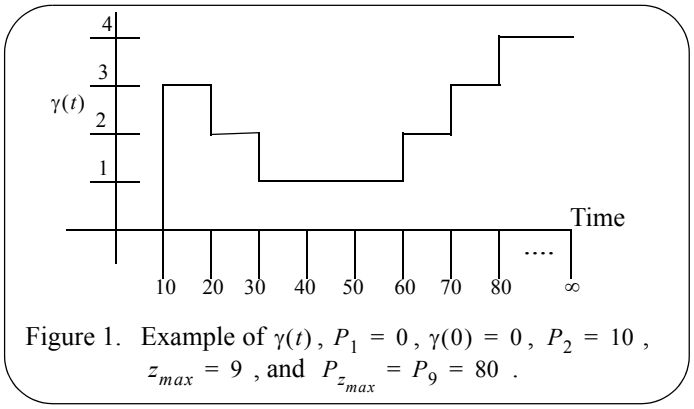


Figure 1. Example of $\gamma(t)$, $P_1 = 0$, $\gamma(0) = 0$, $P_2 = 10$, $z_{max} = 9$, and $P_{z_{max}} = P_9 = 80$.

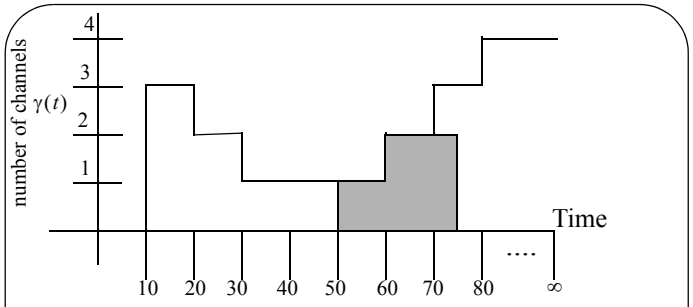


Figure 2. Shaded area shows the allocation for the example 5GB file transfer, with $T^i_{req} = 50$ and $R^i_{max} = 2$ channels. Per-channel link capacity is 10Gbps and one time unit is 100ms.

3.3 Simulation comparison of VBLS against FBLS and PS

In this section, we evaluate the performance of VBLS via simulation. The performance metric of interest here is **normalized delay,** denoted *D*, which is defined as follows:

.

$$D = \frac{\sum_{i=1}^{n} (F^i \cdot d_i)}{\sum_{i=1}^{n} F^i} \qquad \text{EQ(1)}$$

where F^i is the file size of the i^{th} transfer, d_i is the file transfer delay of file i, and n is the total number of transfers. The file transfer delay d_i is defined as the time duration between T^i_{req} and the instant when the transmission of file i is complete.

The primary objective of our simulation study is to compare the performance of VBLS with that of two alternative file transfer schemes serving the same file requests: (1) a **Packet-Switched (PS)** system and (2) a **Fixed-Bandwidth List Scheduling (FBLS)** scheme. FBLS is the greedy scheme that schedules each file request to start as soon as possible, using a *fixed* bandwidth of R^i_{max}. The rationale for this comparison is to illustrate that although VBLS is a circuit-based resource sharing scheme, its delay behavior (on a file-by-file basis) more closely mimics packet switching than it does FBLS. As pointed out before, standard circuit switching using FBLS is expected to produce significantly higher normalized delay than packet switching, simply because ongoing file transfers cannot exploit the release of bandwidth resulting from completed file transfers. However, the variable-bandwidth nature of VBLS in scheduling file transfers mitigates this higher delay; indeed, by design, VBLS exploits the bandwidth released by completed file transfers.

3.3.1 Basic simulation setup

We assume that file transfer requests arrive according to a Poisson process with rate λ based on the findings in [24]. The requested start time T^i_{req} for all transfers is assumed to be equal to the corresponding call arrival times, i.e., all calls are of the "immediate-request" type. We assume that file sizes are distributed according to a bounded Pareto distribution [25]. Specifically, the file size probability density function is given by:

$$f_X(x) = \frac{\alpha k^\alpha x^{-\alpha-1}}{1-\left(\frac{k}{p}\right)^\alpha}, \; k \le x \le p \qquad \text{EQ(2)}$$

where α is the shape parameter, and k and p are the lower and upper bounds, respectively, of the allowed file-size range.

In the simulation of the packet-switched system, files are divided into packets of length 1500 bytes and arrive at the infinite

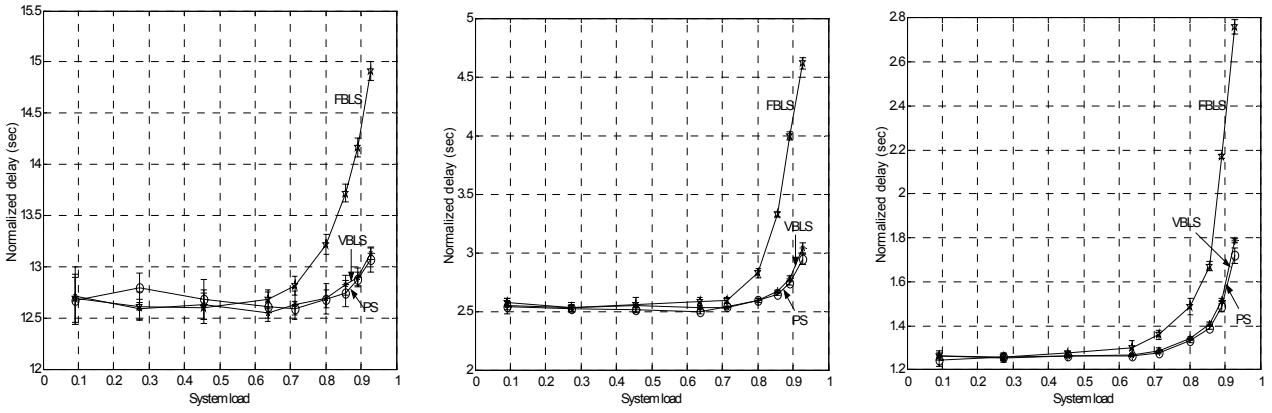
packet buffer at a constant packet rate equal to R_{max}^{i} divided by the packet length. In other words, the packet interarrival time for a file is the packet length divided by the R_{max}^{i} value for the file.

For the input parameters, we choose $\alpha = 1.1$, $k = 500$ MB, and $p = 100$ GB. Instead of a constant R_{max}^{i} for all calls, here we allow three values for R_{max}^{i}: 1, 5, and 10 channels with corresponding probabilities of 0.3, 0.3, and 0.4, respectively. The link capacity C is 100 channels. The bandwidth of each channel is 10Gbps.

3.3.2 Numerical results

In our simulation study, we measure the normalized delays over different values of the file arrival rate λ. We then plot the measured normalized delays versus **system load**, defined as λ multiplied by the mean file size divided by the link capacity. Note that for stability the system load must be below 1.

Figure 3 shows plots of the normalized delays (in seconds) versus system load for VBLS, FBLS, and PS. Our plots are categorized according to the value of R_{max}^{i}; the rationale here is that normalized delays (especially at low loads) are naturally limited by their R_{max}^{i} values, and so comparing delays for files with different R_{max}^{i} values is inappropriate.



(a) R_{max}^{i} of 1 channel (b) R_{max}^{i} of 5 channels (c) R_{max}^{i} of 10 channels

Figure 3. A comparison of FBLS, PS, and VBLS normalized delay for files requesting different R_{max}^{i}

As we can see in Figure 3, VBLS achieves delay values that lie below that of FBLS, as expected. Even more important, the delay performance of VBLS is indistinguishable from packet switching. This serves to illustrate our main point — that by taking into account file sizes and varying the bandwidth allocation for each transfer over its transfer duration, we mitigate the performance degradation usually associated with circuit-based methods.

We note that our simulation of the PS scheme is of an infinite-buffer system. This is clearly an idealized packet-switching scenario. In practice, buffers will be finite, which means packet losses will occur due to congestion. Mechanisms such as TCP's congestion-control schemes are then required to recover from these packet losses with retransmissions and rate adjustments. TCP mechanisms can add significant delays to the total file transfer delays [26]. We plan to compare our VBLS scheme with a PS scheme that employs TCP (standard or improved versions) for a more realistic comparison in a subsequent paper.

In comparing VBLS to FBLS, we see from Figure 3 that the performance gains are significant. However, the penalty paid by VBLS is in the added complexity of switches. First a switch needs to maintain the available bandwidth as a time-varying function for all its interfaces, unlike current-day TDM/FDM switches that only maintain current available-bandwidth information. Second, switches will need timer mechanisms to reconfigure the crossconnections at time-range boundaries for all ongoing connections.

3.4 Practical considerations

VBLS achieves close-to-PS performance at the cost of complexity relative to FBLS. First, as noted in Section 1, VBLS requires that the circuit switches be reprogrammed multiple times within a transfer unlike in the fixed-bandwidth allocation mode where the switch is only programmed at the start and end of a call. With electronic TDM switches, where switch programming times are in the order of nanoseconds, impact of reprogramming on utilization will be less than with optical Micro-Electro-Mechanical Switches (MEMS), in which switch programming times are in the order of milliseconds. If VBLS is used only for very large files, then these overheads will be relatively insignificant.

Second, to make it practical to implement $\gamma(t)$, the capacity availability function, we need to limit the number of bandwidth change points z_{max}. For this purpose, we discretize time and only allow for bandwidth changes to fall on discrete time instances. The smaller the discrete time unit, the larger the storage needed for $\gamma(t)$. The larger this unit, the worse the utilization because bandwidth cannot be reassigned in the middle of a time range. Details such as these will be explored in implementations of VBLS.

Propagation delays and clock synchronization issues become important in distributed implementations of this scheduling algorithm. For now, since the dynamic provisioning of lambdas is handled in a centralized manner, the VBLS scheduler can also be centralized avoiding propagation and clock synchronization problems. Maintenance of the schedules for switch reprogramming should ideally be at the switches themselves with timer-based triggers.

4. Varying-Bandwidth Transport Protocol (VBTP)

To design a transport protocol for lambdas (end-to-end circuits) scheduled by VBLS, we start by considering the purpose and role of transport protocols. Transport protocols perform the functions necessary to achieve reliable transfer of data on an end-to-end basis. The end-to-end paths in these LambdaGrids, as we envision them, will consist of GbE or 10GbE segments at the ends connected via wide-area SONET/SDH circuits and/or WDM all-optical lightpaths. Since resources are reserved in a dedicated manner for these circuits, there is no contention for resources during the actual data transfers, and hence, no possibility of data loss at the circuit-based network switches unlike in packet-based network switches. Nevertheless losses can occur even on these lambdas due to (i) **link errors** and (ii) **receive-buffer overflows**. Link errors arise from bit and burst errors on the physical media. Even though optical fiber, the physical medium of Lambda networks, is fairly reliable, link errors are unavoidable. We will consider error-control solutions for link-error recovery.

Next consider receive-buffer overflows. What are these and why do they occur? At first glance, it appears that with a dedicated circuit, one could simply match the sending rate to the receiver rate (which is ideally equal to the circuit rate), thus eliminating the need for receiver buffers. For example, in end-to-end telephone circuits, senders (speakers) generate audio data at the same rate at which receivers (listeners) consume the data. However, unlike telephones that perform their single dedicated function, general-purpose computers that are typically involved in file transfers engage in several other activities concurrent with file transfers. This requires operating systems to schedule various tasks in and out of the processor as needed, which implies that data received on a Network Interface Card (NIC) is not moved at a guaranteed constant rate from the NIC to the disk. Furthermore disk access rates are not constant. There can be significant variability based on the location to which data needs to be written. Variability also arises at the sender. Based on when the operating system at the sender schedules the network-related kernel threads and drivers, data is moved from disk to memory (user-space and/or kernel-space) to the NIC at varying rates. This variability at the receiver and the sender leads to the interesting question of how to select an appropriate R^i_{max}. If a pessimistic rate is chosen to avoid (at all costs) the possibility of the sender sending data faster than the rate at which the receiver can move the data, the overall transfer delay could be higher than if an optimistic rate is chosen allowing for losses and subsequent retransmissions. Thus rate-based, flow-control schemes are not trivial to implement.

To understand rate-based flow control, we experiment with UDP-based, transport-protocol implementations, such as SABUL [14]. We experimented with other protocols such as Tsunami, RDUDP and UDT, but obtained fairly similar results.

Due to a lack of space, and given that our goal is to use these implementations to experiment with rate-based flow control rather than to compare different protocols, we only include our results from the SABUL experiments.

Section 4.1 describes considerations for flow control along with the results of our experiments with SABUL. Section 4.2 describes error-control mechanisms to handle both link errors and losses due to receive-buffer overflows. Finally, in Section 4.3, we discuss effects induced by the use of VBLS rather than a fixed-bandwidth allocation scheme.

4.1 VBTP flow control and experiments with SABUL

There are three well-known, flow-control methods: ON/OFF, window-based, and rate-based. In ON/OFF and window-based flow control schemes, the receiver sends messages to control the behavior of the sender dynamically. These receiver-based flow-control schemes are suitable for networks where the available bandwidth and receiver rates are unknown to the sender, such as the Internet. However, when used in circuit-switched networks, they leave open the possibility of the circuit lying idle while a sender awaits an ON or "window-open" signal from the receiver, which could lead to poor circuit utilization. In VBLS, since the circuit rate assignment is known a priori to the sender, it is possible to set the sending rate to match the circuit rate/receiver rate. Therefore, the flow-control scheme in VBTP should be rate-based. Rate-based flow control uses the available bandwidth in an efficient manner. It can be implemented by setting the inter-packet generation time at the sender.

SABUL is a rate-based protocol designed for use on the Internet where the sender senses the available bandwidth and adjusts its sending rate accordingly. SABUL is implemented on top of UDP in user space (i.e., not in kernel space). It uses UDP to transfer data and TCP to send control messages. The rate-based algorithm, which tunes the inter-packet transmission time, addresses both the flow control (receive-buffer overflows) and congestion control (network switch buffer overflows) problems. By disabling automatic rate adjusting within a transfer, we can study the impact of sending rates on receive-buffer overflows.

In our experiments, we connected two DELL Precision 650 (P650) workstations via a direct 1Gbps Ethernet link. Each host has a 2.4-GHz Intel Xeon™ CPU connected to a 533-MHz front-side bus (34Gbps CPU bandwidth), an E7505 chipset with 512MB of DDR 266MHz memory (17Gbps memory bandwidth), an 80GB ATA/100 7200 RPM EIDE disk drive (55MB/s or 440Mbps average access rate measured by Diskspeed [27]), and a 64bit/100MHz PCIx bus for the GbE NIC (6.4Gbps network bandwidth). The operating system on both hosts is RedHat Linux 9 with version 2.4.20 kernel.

We ran SABUL code on both the hosts and transferred a 725MB file between them. We began the experiment with default

SABUL settings: 256KB UDP buffer size and 1500B MTU. SABUL's congestion control is disabled in our experiments by setting the same value for the initial rate, the maximum rate, and the minimum rate. The sending rate was fixed during each transfer[1].

Figure 4 plots transfer throughput and packet loss versus sending rate. As the sending rate is increased from a low value, the plots show the throughput increasing as expected with zero or a small packet-loss rate. When the sending rate is increased to a certain level (~200Mbps in Figure 4), the packet loss becomes significant. This happens because of the limitations of the end-host hardware and software involved in moving blocks of the file from disk into application memory, kernel memory and finally into the NIC at the sender, and similarly through the NIC, kernel memory, application memory and disk at the receiver. Further increase of the sending rate beyond 200Mbps leads to excessive errors, causing the retransmissions to impact the overall throughput. As a result, the throughput slowly reaches an "optimal" value (~400Mbps in Figure 4) at a 570Mbps sending rate and then drops down. This "optimal" value matches closely the 440 Mbps disk access rate[2], the expected bottleneck rate in the experiment. The results from the memory-to-memory transfer experiments are also shown in Figure 4. As expected, by removing the effects of disk access, which is the major bottleneck in the disk-to-disk transfer, we can achieve a higher throughput (up to 910Mbps) without incurring too much packet losses.

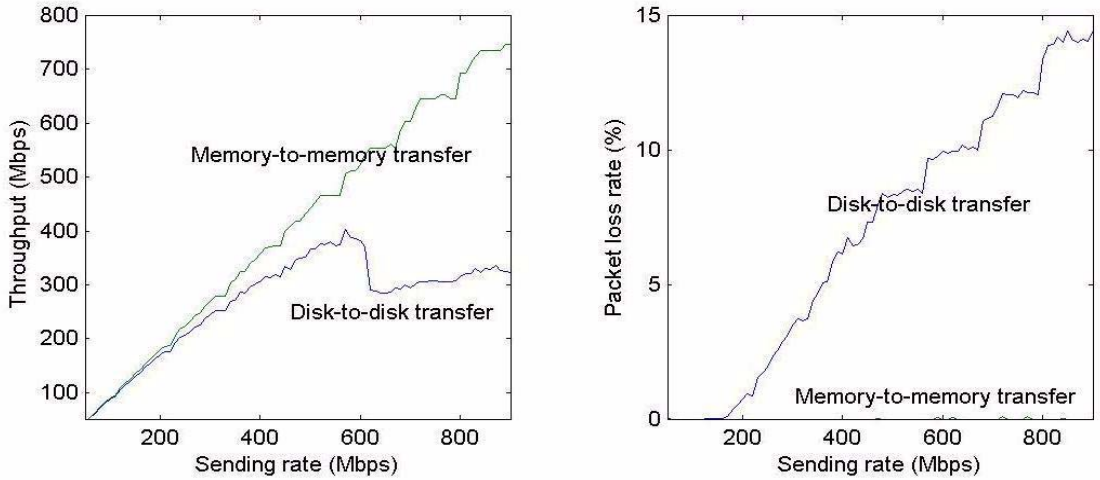


Figure 4. Plots of throughput and packet loss rate vs. the sending rate in SABUL experiments (MTU=1500B, UDP buffer size=256KB, SABUL data block size=7.34MB)

The UDP buffer size has a large impact on the SABUL's performance. We could achieve a better performance with a larger UDP buffer size. A larger UDP buffer reduces the possibility of receive-buffer overflows due to variations in the receiving rate, and therefore alleviates its adverse impact on the throughput. In the experiment, we set the UDP buffer size to be four

times larger than the default setting (256KB). With a 1MB UDP buffer, the average throughput at a 500Mbps sending rate increases to 392Mbps (a 6.8% improvement from original 367Mbps in Figure 4) and the loss rate drops from 8.35% to 0.14%. A larger MTU also tends to improve the throughput, but the improvement is not significant. For example, with a 9000-byte MTU and a default UDP buffer (256KB), the average throughput achieved at a 500Mbps sending rate is 374Mbps (only a 2% improvement from original 367Mbps in Figure 4) and the loss rate is at the same level as with the default setting.

From the experiments, we learn that packet losses resulting from the unpredictability of network related processing at the end hosts are hard to avoid. To achieve the best transfer rate, the R_{max}^{i} setting should be higher than the pessimistic zero-loss rate, which means that some losses will occur due to "receive-buffer overflows" and will need to be recovered through retransmissions.

4.2 VBTP error control

To counter losses from link errors and receive-buffer overflows, we propose to use the selective-ARQ (selective automatic repeat request) scheme to achieve a high efficiency. Negative acknowledgements (NACKs) can be used to indicate packet losses instead of requiring the sender to maintain timers and await positive acknowledgements (ACKs) because of the guaranteed in-sequence delivery of data blocks on dedicated circuits. However, positive ACKs are still needed because the sender needs to update its retransmission buffers. Retransmission buffers are required at the sender because of the high delay overheads involved in disk accesses. Ideally, if one could implement VBTP in an OS-bypass technique where a software program on a processor on the NIC or hardware circuitry moves data directly from the disk to the NIC for transmission, then no retransmission buffers will be needed at the sender. If a block of data is lost, it can re-extracted directly from disk and retransmitted. However, in software implementations on general-purpose hosts using standard Ethernet NICs, we will need retransmission buffers at the sender where data is held until acknowledged to avoid repeated disk accesses. Retransmission buffers cannot be too large because of the limited memory size at end hosts. Therefore we need ACKs to confirm the delivery of packets and allow the release of the corresponding space in the retransmission buffer. There is a trade-off between the retransmission buffer size and the frequency of sending positive-ACKs. The longer the time interval between two consecutive ACKs, the less the overhead incurred. But the retransmission buffer has to be correspondingly larger to store more unacknowledged packets. We need further experimentation to select appropriate values for the retransmission buffer size.

Resequencing buffers are similarly needed at the receiver to accumulate together Ethernet frames to create a block before performing a write access of the disk[3]. Accessing the disk for each Ethernet frame will result in excess overhead. Again, with

an OS-bypass implementation, resequencing buffers can be eliminated.

As described in Section 1, the system architecture allows for dual paths: (i) a dedicated end-to-end high-speed circuit and (ii) a TCP/IP path. The question then is whether to send the retransmissions on the dedicated end-to-end circuit or on the TCP/IP path. Our answer is to use the dedicated end-to-end circuits for retransmissions unless these are needed at the end of the transfer. The reason for not wanting to use the dedicated circuit for any retransmissions needed at the end of a transfer is that the circuit will have to lie idle while the sender awaits acknowledgment of data reception from the receiver. On the other hand, if an aggressive R^i_{max} is used, we can expect a fair number of retransmissions to handle receive-buffer overflows. Hence we recommend using the dedicated end-to-end circuits for most of the retransmissions.

This question of how to send the retransmissions is in fact more complicated than the above discussion indicates when considered in the context of VBLS. We address VBLS-induced effects on the transport protocol design in the next sub-section.

4.3 VBLS-induced effects on transport protocol design

First, consider the VBLS requirement of end hosts having to specify file sizes in their scheduling requests. In sections 3.1 and 3.2, we described the VBLS approach for deriving a TRC allocation for a request using the specified file size F^i. However, in the discussion of sections 4.1 and 4.2, we see that link errors and receive-buffer overflows are unavoidable, both of which result in retransmissions. This means the TRC allocation should be determined not just for the initial file transfer but also for retransmissions. While the file size is known, the number of retransmissions needed or the size of retransmitted data cannot be predicted a priori and can, at best, be only characterized statistically.

Second, a problem arises if the sender does not send at exactly the rates specified in the TRC vector[4]. To understand this effect, assume that the receiver is not a bottleneck, which means no losses occur due to receive-buffer overflows. Even in this ideal situation, if the sender does not change its sending rate exactly as indicated in the TRC vector, problems can arise. If the sender sends at a lower rate than the assigned rate, then the transfer will simply not complete by the scheduled end time. If however the sending rate is higher at a burst level than the allocated circuit rate, then losses will occur at the Ethernet to Ethernet-over-SONET/SDH/WDM mapping network entity. For example, if a host has a 1Gbps Ethernet NIC but we allocate a lower rate wide-area circuit (say if network resources are not available or the receiver cannot handled 1Gbps), then, even if the VBTP implementation sends data at the lower rate of the circuit, the Ethernet driver could pump out frames in a burst at the higher 1Gbps rate.

To solve above problems, we propose adding a margin to the file size F^i when the VBLS scheduler computes a TRC allocation. For example, if we expect a 10% packet loss rate, then the file transfer size F^i specified in the transfer request should be set to 110% of the actual file size. If the actual file transfer is finished before this "over-scheduled" end time, the circuit should be released immediately to allow for a reuse of resources. If the actual file transfer does not complete even with this "over-scheduled" time, the circuit should be released and the remaining data transferred on the TCP/IP path. Since the circuit is already over-scheduled, the remaining data should not be too large, which implies that transferring this data on the TCP/IP path should not affect the throughput severely. The proper margin value must be chosen based on a good estimate of the fluctuation of the sending rate and the receiving rate. Overestimation impacts overall system performance, while underestimation impacts individual transfer delays.

It appears that an OS-bypass implementation in which a TDM-like allocation of computing resources can be made to the transport protocol, both at the sender and receiver, is a better match to circuits than a generic OS-based transport-protocol implementation.

5. Conclusions and future work

For scheduling lambdas in a LambdaGrid for file transfers, we proposed a heuristic called Varying-Bandwidth List Scheduling (VBLS). By having end-host applications provide their file sizes to the VBLS scheduler, it is able to make a Time-Range-Capacity (TRC) vector allocation for transfers. This approach overcomes a well-known drawback of using circuits for file transfers in which with a fixed-bandwidth allocation mode fails to allow users to take advantage of bandwidth that becomes available subsequent to the start of a transfer. We demonstrate through simulations that with VBLS we can improve performance over fixed-bandwidth schemes significantly for file transfers. Furthermore, the normalized delay performance of VBLS is indistinguishable from that of PS. Having demonstrated that it is possible to overcome the drawbacks of circuit switching relative to packet switching, we are now in a position to pursue more sophisticated circuit-switching schemes for file transfers. Circuit switching provides a distinct advantage over packet switching in the ease of implementation and management of pricing mechanisms in the allocation of resources. Such services are not easily embodied in packet-switching mechanisms.

We have identified the key features needed in a transport protocol that works in conjunction with VBLS. We call this protocol Varying Bandwidth Transport Protocol (VBTP), one that is a rate-based, flow-control scheme along with a selective-ARQ based, error-control scheme. In close, we identified the impact of transport problems on VBLS scheduling, such as the need to

over-schedule for retransmissions and variations in the sending/receiving rate.

As part of our future work, we plan to include a second class of user requests for lambdas, specifically targeted at interactive applications such as remote visualization and simulation steering. Such a request i will be specified as $(H^i, R^i_{min}, R^i_{max}, T^i_{req})$, where H^i is the holding time for the lightpath, R^i_{min} and R^i_{max} are the minimum and maximum bandwidth acceptable to the user, and T^i_{req} is the requested start time. The VBLS scheduler can handle such requests in the same manner as it does file transfer requests whereby it allocates a TRC vector starting at some time $T^i_{start} \geq T^i_{req}$ and ending at $T^i_{start} + H^i$. During this interval $T^i_{start} \leq t \leq T^i_{start} + H^i$, varying levels of bandwidth will be allocated in a TRC vector such that the capacity assigned in any time range k is not less than R^i_{min} and not greater than R^i_{max}.

1. With VBLS, the sending rate changes within a transfer, but within a time range it is held constant. Our goal with these SABUL experiments is to determine the impact of end host processing variability on fixed-rate transmissions, independent of whether this rate is held for a time range within a transfer or for the entire transfer duration.

2. The stable value is slightly lower than the hard driver's access rate because of the additional processing overhead for packet losses and retransmissions.

3. Again, with an OS-bypass implementation, resequencing buffers can be eliminated.

4. This is in addition to the clock synchronization problem noted in Section 3.4, where the TRC allocation should be interpreted correctly in the sending host's clock.

REFERENCES

[1] M. D. Brown, "Blueprint for the future of high-performance networking introduction," *Communications of ACM*, Vol. 46, No. 11, pp. 30-33, Nov. 2003.

[2] T. DeFanti, C. D. Laat, J. Mambretti, K. Neggers, B. St. Arnaud, "TransLight: a global-scale LamdaGrid for e-science," *Communications of ACM*, Vol. 46, No. 11, pp. 34-41, Nov. 2003.

[3] H. B. Newman, M. H. Ellisman, J. A. Orcutt, "Data-intensive e-science frontier research," *Communications of ACM*, Vol. 46, No. 11, pp. 68-77, Nov. 2003.

[4] D. Bertsekas and R. Gallager, *Data Networks*. Prentice Hall: New Jersey, 1986.

[5] Derek Wright, "Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with Condor", Conference on Linux Clusters: The HPC Revolution, June, 2001, Champaign - Urbana, IL.

[6] M. Pinedo, *Scheduling: Theory, algorithms and systems*. Prentice-Hall: New Jersey, 1995.

[7] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," Intl Workshop on Quality of Service, 1999.

[8] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, A. S. Lapaugh, "Scheduling File Transfers," *SIAM Journal of Computing*, Vol. 14, No. 3, pp. 744-780, Aug. 1985.

[9] T. Erlebach and K. Jansen, "Off-line and On-line Call Scheduling in Stars and Trees," in *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG '97, LNCS1335*, Springer-Verlag 1997, pp. 195-213.

[10] J. T. Havill, W. Mao and R. Simha, "A Lower Bound for On-line File Transfer Routing and Scheduling," in *Proceedings of the 1997 Conference on Information Sciences and Systems*, 1997, pp. 225-230.
[11] D. Wischik and A. Greenberg, "Admission control for booking ahead shared resources," in *Proc. of IEEE Infocom*, San Francisco, CA, Mar. 29 - April 2, 1998, pp. 873-882.
[12] A. Greenberg, R. Srikant, W. Whitt, "Resource sharing for book-ahead and instantaneous-request calls," *IEEE/ACM Transactions on Networking,* Vol. 7, No. 1, pp. 10-22, February 1999.
[13] L. C. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller and H. Wittig, "Issues of reserving resources in advance," in *Proc. of 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV `95)*, Durham, New Hampshire, April 18-21,1995, pp. 27-37.
[14] Y. Gu, X. Hong, M. Mazzucco, and R. L. Grossman, "SABUL: A High Performance Data Transfer Protocol," submitted to *IEEE COMMUNICATIONS LETTERS.*
[15] Y. Gu and R. L. Grossman, "End-to-End Congestion Control for High Performance Data Transfer," submitted to *IEEE/ACM Transaction on Networking*.
[16] Tsunami, http://www.indiana.edu/~anml/anmlresearch.html.
[17] E. He, J. Alimohideen, J. Eliason, N. K. Krishnaprasad, J. Leigh, O. Yu, T. A. DeFanti, "Quanta: A Toolkit for High-Performance Data Delivery over Photonic Networks," *Future Generation Computer Systems*, 1005, 2003.
[18] S.Floyd, "HighSpeed TCP for Large Congestion Windows," February, 2003, *http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-highspeed-01.txt*.
[19] T. Kelly, "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks," presented at PFLDnet 2003, Feb. 3-4, 2003, Geneva, Switzerland, http://datatag.web.cern.ch/datatag/pfldnet2003/.
[20] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, "FAST TCP: From Theory to Experiments," submitted to *IEEE Network*..
[21] I. R. Philip and Y.-L. Liong, "The Scheduled Transfer (ST) Protocol," *3rd Intl. Workshop on Communications, Architecture and Applications for Network-Based Parallel Computing (CANC'99), Lecture Notes in Computer Science,* vol. 1602, Jan. 1999.
[22] IETF, "Remote Direct Data Placement (RDDP)," http://www.ietf.org/html.charters/rddp-charter.html.
[23] L. Brakmo, L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 8, pp. 1465-1480, Oct. 1995.
[24] V. Paxson and S. Floyd, "Wide-area traffic: The failure of Poisson modeling," *IEEE/ACM Trans. Networking*, Vol. 3, pp. 226-244, June, 1995.
[25] M. E. Crovella, M. Harchol-Balter, and C. D. Murta, "Task Assignment in a Distributed System: Improving Performance by Unbalancing Load," BUCS-TR-1997-018, October 31, 1997.
[26] M. Veeraraghavan, X. Zheng, H. Lee, M. Gardner, and W. Feng, "CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture," in *Proc. of Opticomm 2003,* Dallas, TX, Oct. 13-17, 2003, extended version of paper presented at PFLDN2003.
[27] http://home.earthlink.net/~alegr/download/diskspeed.htm.