

# Cloud Computing over Light-trail WDM Core Networks

Prasad Gokhale, Tamal Das, Ashwin Gumaste

Department of CSE, Indian Institute of Technology, Bombay, India.

Email: {pkgokhale, tamaldas}@cse.iitb.ac.in, ashwing@ieee.org

**Abstract:** Light-trail, a spatial sub-wavelength optical grooming solution for WDM networks, facilitates dynamic bandwidth provisioning and optical multicasting - essential for cloud computing services. We evaluate through simulations a novel protocol for cloud computing over light-trails.

© 2010 Optical Society of America.

OCIS codes: 060.4250 Networks; 060.4250 Networks

## 1. Introduction

Cloud computing brings together information technology applications as a service on a shared networking infrastructure and has the potential benefits of significant cost reduction and performance enhancement. The underlying network is typically based on high-speed technologies that need to facilitate dynamic bandwidth allocation, efficient grooming and using a low-cost networking technology platform. Light-trails [1] are an efficient high-speed networking solution that have the ability to meet cloud computing applications. We present an algorithm that binds IT-applications to a light-trail based WDM metro network. This algorithm is based on a two-stage auctioning technique and makes good use of the advantages of wavelength sharing in the optical domain while achieving fast response much desired by IT applications. Simulation results validate the use of the technique and thereby bring out the benefits of light-trails as an infrastructural solution for cloud computing.

Light-trails are defined as a generalization of a lightpath such that multiple nodes can take part in communication along the path. A light-trail is analogous to a wavelength bus, thus exhibiting functionalities of optical multicasting and being able to provide time-differentiated sub-wavelength communication to nodes in the light-trail. A light-trail based network leads to spatial wavelength grooming and enables dynamic bandwidth provisioning. Nodes in a light-trail have properties to support bus functionality. In particular, nodes support, on a per-wavelength basis drop-and-continue, passive add properties as well as an out-of-band control channel that enables arbitration of communication within the bus. The OOB control channel is optically dropped and electronically processed at every node. When a node communicates to another node over a light-trail, it does so by forming a connection. Connections are provisioned over the wavelength bus without any optical switch configuration. When a node communicates over the light-trail, other nodes queue their data in electronic buffers. A light-trail MAC protocol is essential to regulate communication. The light-trail is assumed to be time-slotted and nodes compete for time-slots depending on their service requirements. A centralized arbiter is responsible for communication within the light-trail and grants time-slots to nodes depending on their service requests. Since multiple nodes all-optically share the wavelength bus, they individually are allocated sub-wavelength granularity, and this type of grooming has been defined as spatial grooming due to the multitude of nodes time-sharing the bus. A light-trail based service provisioning protocol, optimized for clouds is proposed and described next.

*Notations:* Let the network topology be defined by an undirected graph  $G(N,L)$ , where  $N$  is the set of nodes and  $L$  the set of fiber-links. Denote the  $i^{\text{th}}$  node by  $N_i$  with processing power  $p(N_i)$ , equipped with resource type  $t(N_i)$ . The capacity of each link is assumed as  $C$ . The universal set of services that the network can provision is given by  $S=\{S_1, S_2, \dots, S_n\}$ . A service  $S_i$  is divisible into several interdependent and atomic tasks. We denote the universal set of tasks by  $T=\{T_1, T_2, \dots, T_m\}$ . Task  $T_i$  is subjected to delay bound  $\delta_i$  and requires  $w(T_i)$  units of resource type  $t(T_i)$ . The delay bound of a service is the cumulative time duration of the critical path in its *task dependency graph (TDG)*. Further the start and end-times of  $T_i$ 's execution are denoted by  $t_s(T_i)$  and  $t_e(T_i)$ , respectively. Thus a given task can be provisioned at a node compatible with the task's resource type requirement. Further, the task currently being executed at node  $N_i$  is denoted by  $e(N_i)$ . The task dependency graph of service  $S_i$  is denoted as the directed graph  $G_i = \overline{T}_i, \overline{E}_i$ , where  $\overline{T}_i$  denotes

the set of tasks that together constitutes  $S_i$ .  $\overline{E}_i$  denotes the set of directed links between interdependent tasks. Each link  $E_i \in \overline{E}_i$  is associated with the amount of data  $\rho_{E_i}$  that needs to be transferred between the tasks that it connects. Fig. 1-2 illustrates the above convention. In Fig. 1, each of the nodes  $N_1, \dots, N_5$  are labeled with a 2-tuple where the first parameter

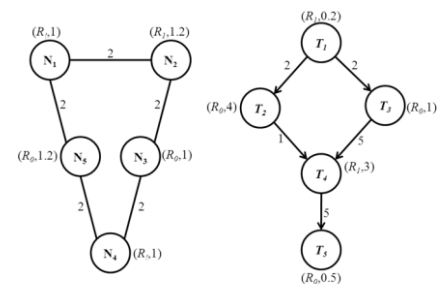


Fig. 1. Sample Network Topology

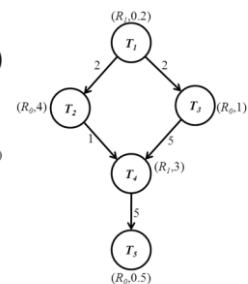


Fig. 2. Task Dependency Graph for a Service

## OWR3.pdf

denotes the resource type ( $R_0$  or  $R_1$ ) and the second parameter denotes the processing power of the corresponding resource. Further, each of the edges is labeled with their respective capacity. Fig. 2 describes the task dependency graph for a service. Each of the tasks  $T_1, \dots, T_5$  is associated with a 2-tuple, *viz.*, required resource type and number of units needed of that resource. The amount of data that needs to be transferred across adjacent tasks is labeled on the respective edges.

*Problem Statement:* The input is the set of services existing at time  $t$  awaiting resources – denoted by  $\Omega(t)$ . The output is an optimal schedule of  $\Omega(t)$  that minimizes total execution time while using the minimal number of light-trails, subject to (1) the capacity constraints of the links and (2) the delay bound of each of the services.

## 2. Cloud Computing over Light-trail Algorithm

Our approach is to prioritize the incoming service requests motivated from the auction-based scheme presented in [2]. The idea is that a service which will be the first to be dropped, if not provisioned, is assigned the highest priority. Further we adapt the virtual light-trail topology growth algorithm [2] for efficient and dynamic provisioning of tasks. Tasks are provisioned on existing light-trails or by dimensioning an existing light-trail or creating a new light-trail. *Algorithm 1* provides the integrated schedule for the problem under consideration. It runs in parallel with *Algorithm 3* and *Algorithm 4*. A detailed description of our algorithm is now presented.

*Convention:* We now discuss the conventions used for our analysis.  $K(t)$  denotes the set of existing light-trails at time  $t$ .  $f_{ij}$  is the flow from node  $N_i$  to node  $N_j$  in the network.  $t_{HP}$  is the time required to setup a light-trail, whereas  $T_S$  denotes each time-slot duration. We denote a light-trail by  $k$  or  $k_i$ .  $\|N_i \oplus N_j\|_k = 1$ , if  $N_i$  is upstream of  $N_j$  on light-trail  $k$ , and 0 otherwise.  $U_k$  denotes the utility (total flow divided by the capacity) of light-trail  $k$ . The upper and lower bound on the utility is denoted by  $UB_{Thresh}$ ,  $LB_{Thresh}$ . The start and end-node of light-trail  $k$  is denoted by  $conv(k)$  and  $end(k)$ . Light-trail  $k$  is established on wavelength  $\lambda_k$ .  $\psi_{ik}(t)$  denotes the maximum duration of time that  $N_i$  can wait for a transmission slot after which its most stringent service packet will be dropped.

<pre> R(t) ← Set of network-wide available resources Ω(t) ← Prioritize Ω(t) using Algorithm 2 while Ω(t) is not empty   S<sub>i</sub> ← Highest priority service of Ω(t)   G<sub>i</sub> = <math>\bar{T}_i, \bar{E}_i</math> ← Task dependency graph for S<sub>i</sub>   while <math>\bar{T}_i</math> is not empty     T<sub>j</sub> ← Pop head (task) of <math>\bar{T}_i</math>     Ψ(T<sub>j</sub>) ← Set of available resources compatible with T<sub>j</sub>     if T<sub>j</sub> is an entry node in G<sub>i</sub>       Assign T<sub>j</sub> to earliest available node N<sub>i</sub> ∈ Ψ T<sub>j</sub>       t<sub>s</sub>(T<sub>j</sub>) ← Earliest time when N<sub>i</sub> is available     else       Assign T<sub>j</sub> to N<sub>j</sub> ∈ Ψ T<sub>j</sub>, such that N<sub>j</sub>'s serves any       predecessors ∃ links from N<sub>j</sub> to most of T<sub>j</sub>'s predecessors       Provision light-trails between each one of T<sub>j</sub>'s       predecessors and N<sub>j</sub> using Algorithm 5       t<sub>s</sub>(T<sub>j</sub>) ← Earliest time when N<sub>j</sub> is available and all       predecessors of T<sub>j</sub> have finished execution and have transmitted the       required data to N<sub>j</sub>     end if     t<sub>e</sub>(T<sub>j</sub>) ← t<sub>s</sub>(T<sub>j</sub>) + w(T<sub>j</sub>)/p(N<sub>j</sub>)     e N<sub>j</sub> ← e N<sub>j</sub> ∪ T<sub>j</sub>     Update R(t)   end while   Ω(t) ← Ω(t) - S<sub>i</sub> end while return the schedule </pre> <p style="text-align: center;"><b>Listing-1: Pseudo code for Algorithm 1</b></p>	<pre> arrivalTime ← Arrival times of services in Ω(t) Δ ← Delay bounds of services in Ω(t) j ← 1, t ← current time instant while (Ω(t) is non-empty)   S<sub>i</sub> ← arg min [ Δ i - t - arrivalTime i ] / Δ i, ∀i   output(j) ← S<sub>i</sub>   Ω(t) ← Ω(t) - S<sub>i</sub>   Re-evaluate arrivalTime, Δ   j ← j + 1 end while return output </pre> <p style="text-align: center;"><b>Listing-2: Pseudo code for Algorithm 2</b></p>
<pre> if (a task has finished its execution and released its resources)   Update R(t)   Reprioritize Ω(t) using Algorithm 2 end if </pre> <p style="text-align: center;"><b>Listing-3: Pseudo code for Algorithm 3</b></p>	<pre> if (new service request has arrived)   Reprioritize Ω(t) using Algorithm 2 end if </pre> <p style="text-align: center;"><b>Listing-4: Pseudo code for Algorithm 4</b></p>
<pre> Φ<sub>j</sub> ← Set of T<sub>j</sub>'s predecessors are disconnected from N<sub>j</sub> for each element N<sub>k</sub> in Φ<sub>j</sub>   if an existing light-trail (LT<sub>p</sub>) connects N<sub>k</sub> and N<sub>j</sub>     then use LT<sub>p</sub> for data transfer between N<sub>k</sub> and N<sub>j</sub>   else if an existing light-trail (LT<sub>p</sub>) includes N<sub>k</sub> (or N<sub>j</sub>) and can     be extended to N<sub>j</sub> (or N<sub>k</sub>)     Extend LT<sub>p</sub> to N<sub>j</sub>   else     Create a new light-trail between N<sub>k</sub> and N<sub>j</sub>   end if   Update L end for </pre> <p style="text-align: center;"><b>Listing-5: Pseudo code for Algorithm 5</b></p>	

*Algorithm 1 (Integrated Scheduling):* This algorithm outlines the steps required to efficiently provision an incoming set of services ( $\Omega(t)$ ) and outputs an optimal schedule for execution of  $\Omega(t)$  (See Listing-1). The services are prioritized using *Algorithm 2* and are provisioned in order of decreasing priority. The constituent tasks of a service are allocated resources in the sequence of their execution based on the present service's TDG. A task is allocated a resource so as to minimize the data transferred by its predecessor tasks over the light-trails.

*Algorithm 2 (Service Prioritization Algorithm):* In this algorithm, we compute the relative priority of different services. Priority of a service is determined using the maximum amount of time that a service can wait before being dropped, normalized over its delay bound. The input to this algorithm is  $\Omega(t)$  whereas the output is  $\Omega(t)$  sorted in descending priority as shown in Listing-2.

*Algorithm 3 (Task completion event handler):* The task completion event handler upon completion of a task releases the resources assigned as depicted in Listing-3.

*Algorithm 4 (Service arrival event handler):* On arrival of a new request a service is added to  $\Omega(t)$  and *Algorithm 2* is invoked to re-prioritize  $\Omega(t)$ . The pseudo code to accomplish this is illustrated in Listing-4.

*Algorithm 5 (Light-trail provisioning):* For a task to begin its execution either it may be processed on the same node as its predecessor(s) or the processed data from all its predecessors should be transferred (before its execution) to the node on which it is provisioned. In the latter case, there must exist light-trail(s) connecting the ingress node(s) to the egress node. To provision a light-trail we first check if an existing light-trail can accommodate the required connection. If not, then we check if any existing light-trail can be dimensioned at either ends to accommodate both the nodes. If not, then we create a new light-trail to provision the data transfer. The inputs to this algorithm are  $T_j$ ,  $N_j$ ,  $G_i$ , whereas the output is light-trails provisioned between  $N_j$  to each of  $T_j$ 's predecessors (See Listing-5).

### 3. Simulation

A discrete event simulator for cloud over ring WDM networks with 12 nodes and 12 wavelengths was developed in Java™. There were 12 types of services with each service having between 4 to 6 tasks. Services arrived with a Poisson distribution and predefined holding time. Light-trail line rate was assumed to be 1Gbps. TDGs were manually given as input to the simulator. Each service had a predefined execution deadline (inclusive of the processing time) between 200 to 460 ms. Load was measured as the sum of arrival rates at all the nodes, though the light-trail was assumed to work at a maximum normalized load less than 10% to show realistic metro environment behavior.

Shown in Fig. 3 is a viewgraph of utilization (ratio of busy to total system time) as a function of load for dependent (with processor in the TDG) and independent services. Note that as the load increases the dependent plot sinks in terms of utilization. This is because the communication between multiple nodes housing the tasks increases significantly. This is also observed in the viewgraph in Fig. 4 where the blocking probability increases as a function of load for the dependent services. The independent services have a linear blocking probability. Shown in Fig. 5 is the average queueing delay of the dependent and independent cloud services over the light-trail network. The sinusoidal behavior is explained as follows: Initially the number of light-trails is less, hence delay increases due to saturation in light-trails. Once total network capacity increases, the delay falls till the load forces the delay to rise again. The number of light-trails then increases and capacity is made available resulting in a fall of the delay. The sinusoidal behavior continues to repeat till the steady state delay reaches the service latency limit.

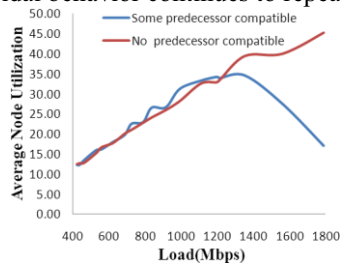


Fig. 3. Node Utilization v/s Load.

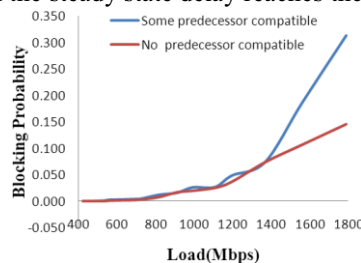


Fig. 4. Blocking Probability v/s Load.

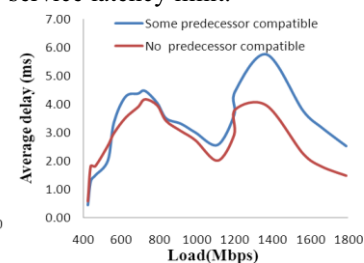


Fig. 5. Delay v/s Load.

### 4. Conclusion

Cloud computing is simulated over the light-trails WDM ring environment. Dependent and independent cloud services are seen to have varied responses over the light-trail infrastructure.

### References

- [1] A. Gumaste and S. Zheng, *IEEE Commun. Mag.* Vol. 21. No 3. pp-21-29.
- [2] A. Gumaste, T. Das, A. Somani and N. Ghani, *IEEE/OSA Journal of Lightwave Technology (Under Major Revision)*