Bandwidth Scheduling and Path Computation Algorithms for Connection-Oriented Networks
Confidential Draft, Not For Circulation

Sartaj Sahni Nageshwara Rao Sanjay Ranka Yan Li
Eun-Sung Jung Nara Kamath [*]

Abstract

There has been an increasing number of network deployments that provide dedicated connections through on-demand and in-advance scheduling in support of high-performance applications. We describe algorithms for scheduling and path computations needed for dedicated bandwidth connections for fixed-slot, highest available bandwidth in a given slot, first available slot, and all-available slots computations. These algorithms for bandwidth scheduling are based on extending the classical breadth-first search, Dijkstra, and Bellman-Ford algorithms. We describe a bandwidth management system for UltraScience Net that incorporates implementations of these algorithms.

Keywords: backend signaling, MPLS/GMPLS distributed implementation, resource scheduling, multi-domain scheduling.

1 Introduction

A number of large-scale science and commercial applications produce large amounts of data, of the order of terabytes to petabytes, that must be transported across wide-area networks. For example, large simulation data sets produced by an eScience application on a supercomputer may be archived at a remote storage site, or bank transaction records or inventories of large department stores may be synchronized during off-peak hours. More generically, when data providers and consumers are geographically distributed, dedicated connections

[*]Mr. Li, Mr. Jung, and Drs. Ranka and Sahni are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 ({yanli,ejung,ranka,sahni}@cise.ufl.edu). Dr. Rao is with Oak Ridge National Labs., Oak Ridge, TN (raons@ornl.gov). Mr. Nara Kamath is with Advanced Algorithms and Systems, VA (nara_kamath@yahoo.com).

are needed to effectively support a variety of remote tasks including data mining, data consolidation and alignment, storage, visualization and analysis [6]. More specifically, dedicated bandwidth channels are critical in these tasks to offer (i) large capacity for massive data transfer operations, and (ii) dynamically stable bandwidth for monitoring and steering operations. It is important that these channels be available when the data is or will be ready to be transferred. Thus, the ability to reserve such dedicated bandwidth connections either on-demand or in-advance is critical to both classes of operations.

The importance of dedicated connection capabilities has been recognized, and several network research projects are currently underway to develop such capabilities. They include User Controlled Light Paths (UCLP) [19], UltraScience Net (USN) [7], Circuit-switched High-speed End-to-End Transport ArcHitecture (CHEETAH) [5], Enlightened [11], Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) [1], Japanese Gigabit Network II [15], Bandwidth on Demand (BoD) on Geant2 network [12], On-demand Secure Circuits and Advance Reservation System (OSCARS) [2] of ESnet, Hybrid Optical and Packet Infrastructure (HOPI) [13], Bandwidth Brokers [8] and others. In addition, production networks at the national and international scale with such capabilities are being deployed by Internet2 [14] and LHCNet[16]. Such deployments are expected to be on the increase and proliferate into both shared and private network infrastructures across the globe in the coming years.

A control plane framework for supporting dedicated channels (Figure 1) consists of the following components [17]: (a) client interface, (b) server front-end, (c) user management, (d) token management, (e) database management, (f) bandwidth scheduler, and (g) signaling daemon. Based on the network topology information and existing link bandwidth allocations, the scheduler computes an appropriate path in response to a specific user request. Upon path computation, bandwidth allocations of each link along that path are updated, and then a signaling daemon invokes appropriate scripts to set up or tear down the connections along the computed or established path, respectively. In this paper, we focus on the algorithms needed for the bandwidth scheduler component. This centralized control plane architecture is similar to that of the bandwidth brokers [9], but the former supports advance bandwidth

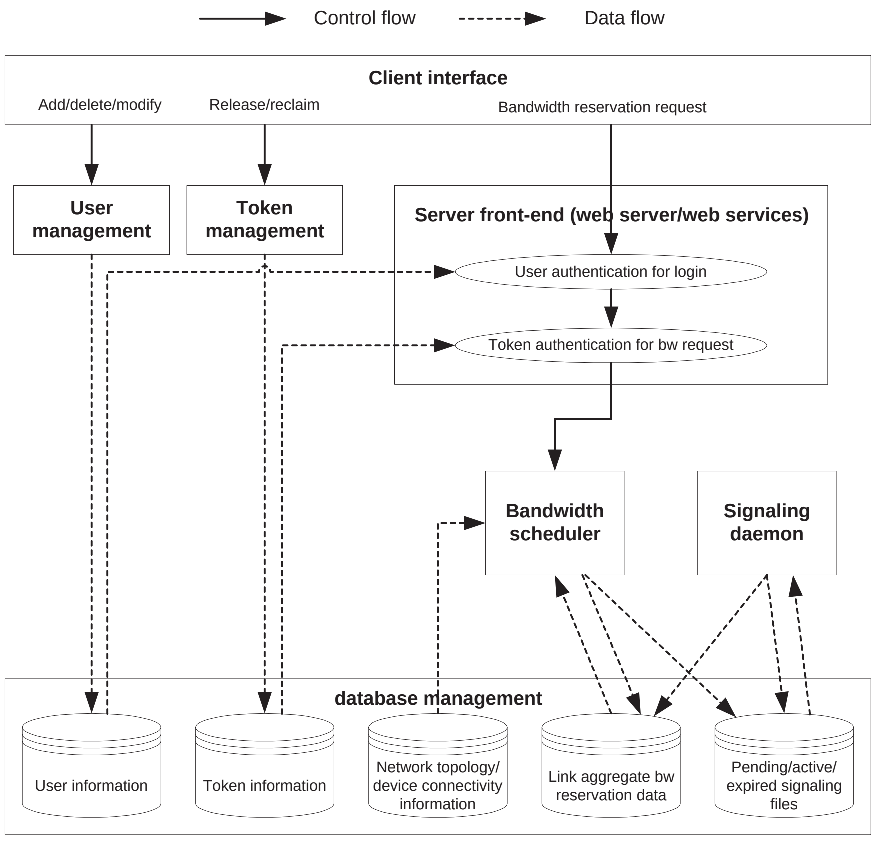

Control flow
Data flow
Client interface
Add/delete/modify
Release/reclaim
Bandwidth reservation request
User management
Token management
Server front-end (web server/web services)
User authentication for login
Token authentication for bw request
Bandwidth scheduler
Signaling daemon
database management
User information
Token information
Network topology/ device connectivity information
Link aggregate bw reservation data
Pending/active/ expired signaling files

Figure 1: Framework of control plane: function components, control flow, and data flow [17].

reservation in addition to the on-demand provisioning of the latter.

The bandwidth scheduling and path computation algorithms allocate the available bandwidth to the connections in time and space. There are two basic modes in which dedicated connections are provisioned based on the current availability of bandwidth on the needed constituent links:

(a) In **on-demand** mode a connection request is made when needed, and it is then accepted or denied depending on the current bandwidth availability; and

(b) In **in-advance** mode, a connection request is granted for future time-slots based on bandwidth allocation schedules.

A majority of current networks offer on-demand capabilities, which are typically supported by Multiple Protocol Label Switching (MPLS) [10, 4] at layer-3 networks and by Generalized

MPLS (GMPLS) [3] at layer-2 and layer-1. CHEETAH, DRAGON, HOPI, UCLP and JGN offer on-demand provisioning, and GeantII, OSCARS, USN and Enlighten offer in-advance provisioning.

Here we consider that bandwidth on certain links of a network are to be allocated to support dedicated connections; all links may be allocated as in USN or CHEETAH or only certain links on traceroute paths may be allocated as in OSCARS. These links constitute a graph, and a dedicated connection may be composed by concatenating the links and time-slots on them. Our focus here is on bandwidth scheduling and path computation algorithms (BSPCA) that can be applied across a number of connection-oriented networks. In particular, our interest is in provably correct polynomial-time algorithms that compose paths by suitably aligning bandwidth allocations on individual links. To our knowledge, USN [17] has the first implementation of a control plane with mathematically-validated advance scheduling capability. Since advance scheduling is not a part of MPLS/GMPLS specification, this capability is not currently supported by GMPLS-based networks such as CHEETAH and DRAGON. While advance reservation is supported by OSCARS [2] over ESnet, OSCARS' underlying path computation limits connections over links returned by traceroute; thus, it does not explore all available bandwidth inside the network. The scheduler of Enlighten considers allocations over a set of manually provided paths; this approach becomes less practical in large networks where the number of paths to be listed could be quite large.

In Section 2, we describe algorithms for fixed-slot, highest available bandwidth in a given slot, first available slot, all available slots between a pair of vertices, and all-available slots between all pairs of vertices computations. In Section 3, we describe a graphical user interface to input a network topology and to display bandwidth allocations computed by our algorithms, and in Section 4 we briefly describe the integration of these algorithms into USN control plane.

2 Scheduling Algorithms

We consider BSPCAs to compute a dedicated channel from a given source to a given destination for: (i) a specified bandwidth in a specified time slot, (ii) highest available bandwidth in a specified time slot, (iii) earliest available time with a specified bandwidth and duration, and (iv) all available time slots with a specified bandwidth and duration. All these algorithms are mathematically guaranteed to return the correct answers. The first and third algorithms are an extension of the classical breadth-first algorithm [18]; the second is an extension of Dijkstra's shortest path algorithm [18]; and the last is an extension of the Bellman-Ford algorithm [18]. The extended Bellman-Ford algorithm for (iv) is an improvement over the transitive closure algorithm described in [7] in terms of time complexity, and is also better suited for distributed implementation. We also consider the problem of determining all available slots between all pairs of vertices.

A network is represented as a graph $G = (V, E)$ where each node represents a device such as a switch for layers 1-2 and router for layer 3, and each edge represents a link such as SONET or Ethernet. For each edge $e \in E$, we have a list of bandwidth reservations specified as a piecewise constant function of time. In terms of data structure, G is represented using the cost-array-adjacency-list representation [18], and each edge has a list, the TB list, of time-bandwidth pairs associated with it. Let (t_1, b_1), ..., (t_k, b_k) be the TB list associated with edge (u, v). The time-bandwidth pairs (t_i, b_i) are in ascending order of t_i. The pair (t_i, b_i) is interpreted to mean that edge (u, v) has bandwidth b_i available from time t_i to time t_{i+1} $(t_{k+1} = \infty)$.

2.1 Fixed Slot

The task here is to find a path from a source vertex s to a destination vertex d; the path must have an available bandwidth of at least b from a prespecified start time t_{start} to the time t_{end}. Such a path is called a *feasible path.* Although either of the traditional graph search methods–breadth-first and depth-first–could be extended to accomplish this task, we extend breadth-first search because a breadth-first search finds a path from s to d with the fewest number of hops. The extended breadth-first search begins at vertex s and terminates either when vertex d is reached via a feasible path or when it is determined that there is no feasible path to d (Figure 2). In case a feasible path exists, the reverse of the sequence d, $prev(d)$, $prev(prev(d))$, ..., s is one such path. The complexity of the extended breadth first

```
ExtendedBreadthFirstSearch(s,d,prev)
{
    Label vertex $s$ as reached.
    Initialize $Q$ to be a queue with only $s$ in it.
    while ($Q$ is not empty)
    {
        Delete a vertex $w$ from the queue.
        Let $u$ be a vertex (if any) adjacent from $w$.
        while ($u \neq null$)
        {
            if ($u$ has not been labeled and edge $(w, u)$ has bandwidth
            $b$ or more available from time $t_{start}$ to $t_{end}$)
            {
                $prev[u] = w$;
                if ($u == d$) return;
                Label $u$ as reached.
                Add $u$ to the queue.
            }
            $u$ = next vertex that is adjacent from $w$.
        }
    }
}
```

Figure 2: Extended breadth-first-search

search algorithm is $O(n + L)$, where n is the number of vertices in the network and L is the sum of the lengths of the TB lists.

2.2 Largest Bandwidth in a Slot

This computation is achieved by modifying Dijkstra's shortest path algorithm [18] as shown in Figure 3. Here, $b[u][v]$ is the minimum bandwidth available on the edge (u, v) during the specified interval/slot and $bw[u]$ is the maximum bandwidth along paths from the source s to vertex u under the constraint that these paths go through only those vertices to which a maximum bandwidth path has been found already. The complexity of the algorithm is $O(n^2)$ for a general n vertex graph. However, practical network graphs have $O(n)$ edges and the complexity becomes $O(n \log n)$ when a max heap (for example) is used to maintain the bw values.

```
MaxBandwidth(s,d,prev)
{
    $bw[i] = b[s][i]$, $1 \leq i \leq n$.
    $prev[i] = s$, $1 \leq i \leq n$.
    $prev[s] = 0$.
    Initialize L to be a list with all vertices other than $s$.
    for ($i = 1$, $i < n - 1$, $i++$)
    {
        Delete a vertex w from $L$ with maximum $bw$.
        if ($w == d$) return.
        for (each $u$ adjacent from $w$)
            if ($bw[u] < \min\{bw[w], b[w][u]\}$)
            {
                $bw[u] = \min\{bw[w], b[w][u]\}$.
                $prev[u] = w$.
            }
    }
}
```

Figure 3: Modified Dijkstra's algorithm

2.3 First Slot

We define, for each edge (u, v) of the graph, an ST (start time) list $ST(u, v)$, which is comprised of pairs of the form (a, b), $a \leq b$. The pair (a, b) has the interpretation: for every start time $x \in [a, b]$, edge (u, v) has an available bandwidth of at least b from time x to time $x + T$. The pairs on an ST list are assumed to be disjoint and in ascending order. The ST lists may be constructed from the TB lists in $O(L)$ time. Let $a_1 < a_2 < ... < a_q$ be the distinct a values in the (a, b) pairs in all ST lists. It is easy to see that the start time for the first slot with bandwidth b and duration t (if such a slot exists) must be one of these a_is. Therefore, we may find the first slot by determining the smallest a_i for which there is an s to d path such that a_i is in an (a, b) interval of every edge on the path. The algorithm of Figure 4 just does this. In this algorithm $NewBFS$ does a breadth-first search beginning at s. The search uses only those edges that include a_i in one of their (a, b) intervals. The search returns **true** iff a path from s to d is found. The array $prev$ has the same significance as in the extended breadth-first search algorithm of Figure 2.

```
FirstSlot(s,d,prev)
{
    for ($i = 1$, $i \leq q$, i++)
        if ($NewBFS(s, d, a_i, prev)$) return $a_i$;
    return $-1$;
}
```

Figure 4: First slot algorithm

Since the a_is are tried in ascending order and the (a, b) pairs in the ST lists are also in ascending order, it is possible to implement $NewBFS$ so that the total running time of $FirstSlot$ is $O(eq)$, where e is the number of edges in the graph. Since $e = O(n)$ for real-world networks, the complexity is $O(nq)$. The actual s to d path for the first slot may be constructed in $O(n)$ additional time using the $prev$ values as described for the fixed slot algorithm.

2.4 First Slot and All-Available Slots

The concept of an ST list for an edge may be extended to that of an ST list for a path. Let $st(k, u)$ be the union of the ST lists for all paths from vertex s to vertex u that have at most k edges on them. Clearly, $st(0, u) = \emptyset$ for $u \neq s$ and $st(0, s) = [0, \infty]$. Also, $st(1, u) = ST(s, u)$ for $u \neq s$ and $st(1, s) = st(0, s)$. For $k > 1$ (actually also for $k = 1$), we obtain the following recurrence

$$st(k, u) = st(k-1, u) \cup \{\cup_{v \text{ such that } (v,u) \text{ is an edge}} \{st(k-1, v) \cap ST(v, u)\}\} \quad (1)$$

where $\cup$ and $\cap$ are list union and intersection operations. For an n-vertex graph, $st(n-1, d)$ gives the start times of all feasible paths from s to d. The Bellman-Ford algorithm [18] may be extended to compute $st(n-1, d)$. This extension not only allows us to determine all available slots from s to d but gives us also an alternative algorithm for the first slot problem as the earliest start time for a bandwidth b duration t path from s to d is the a value of the first (a, b) pair in $st(n-1, d)$.

It is easy to see that the computation of the $st(*, *)$s may be done in place (i.e., $st(k, u)$ overwriting $st(k-1, u)$) and the computation of the sts terminated when $st(k-1, u) =$

```
algorithm ExtendedBellmanFord(s,d)
{
    initialize $st(*) = st(0,*)$;
    // compute $st(*) = st(n-1,*)$
    put the source vertex into $list1$;
    for (int k = 1; k < n; k++)
    {
        // see if there are vertices whose $st$ value has changed
        if ($list1$ is empty) break;  // no such vertex
        while ($list1$ is not empty)
        {
            delete a vertex $v$ from $list1$;
            for (each edge $(v,u)$)
            {
                $st(u) = st(u) \cup \{st(v) \cap ST(v,u)\}$;
                if ($st(u)$ has changed and $u$ is not on $list2$) add $u$ to $list2$;
            }
            $list1 = list2$;
            make $list2$ empty;
        }
    }
}
```

Figure 5: Extended Bellman-Ford algorithm

$st(k,u)$ for all u. These observations result in the pseudocode of Figure 5 to compute $st(n-1,*)$.

Each iteration of the **for** loop takes $O(l)$ time, where l is the length of the longest st list. Since this **for** loop is iterated a total of $O(ne)$ times, the complexity of the extended Bellman-Ford algorithm is $O(nel)$. For real-world networks with $e = O(n)$, this complexity is $O(n^2l)$. The described extended Bellman-Ford algorithm is an early-terminating variant of the Bellman-Ford algorithm given in [17].

When using the extended Bellman-Ford algorithm to solve the first slot problem, we first find the earliest start time t for a feasible path using $ExtendedBellmanFord$. Then, the actual path may be computed using the function for fixed slot with $t_{start} = t$ and $t_{end} = t+T$. In practice, we expect the first slot algorithm of Figure 4 to be faster as we expect $q < nl$ in practice.

```
algorithm ExtendedFloyd()
{
   for (int k = 1; k < n; k++)
      for (int i = 1; i < n; k++)
         for (int j = 1; j < n; k++)
            $st(i,j) = st(i,j) \cup \{st(i,k) \cap st(k,j)\}$;
}
```

Figure 6: Pseudocode for extended Floyd algorithm

2.5 All-Available Slots for All-Pairs

The extended Bellman-Ford algorithm of Figure 5 computes $st(u) = st(n-1, u)$ for a given source vertex s and all u in $O(nel)$ time. $st(u)$ gives the start time of all available slots of duration d and bandwidth b. So, in $O(nel)$, using Figure 5, we are able to determine all available slots from s to every other vertex u (including vertex d). Furthermore, to determine all available slots between all pairs of vertices, we may run the algorithm of Figure 5 n times, once with each vertex as the source vertex s. So, the time needed to determine all slots between all pairs of vertices is $O(n^2el)$. An alternative strategy to determine all available slots between all pairs of vertices is to extend Floyd's all-pairs shortest path algorithm [18] as is done in [7]. Figure 6 gives the resulting extension. Here, $st(u, v)$ is the ST list for paths from u to v. Initially, $st(u, v) = ST(u, v)$. On termination, $st(u, v)$ gives all possible start times for paths from u to v.

The complexity of the extended Floyd algorithm is $O(n^3l)$, where l is the length of the longest st list. Since, the number of edges in a general graph is $O(n^2)$, the worst-case complexity of using the extended Bellman-Ford algorithm to find all available slots between all pairs of vertices is more (by a factor of n) than using the extended Floyd algorithm. However, for our application, the number of edges in a network is $O(n)$ and both algorithms have the same asymptotic complexity. Moreover, since the extended Bellman-Ford algorithm has an early terminating feature, it is expected to perform better than the extended Floyd algorithm when there are short paths (i.e., a small number of edges relative to $n-1$) between pairs of vertices.

3 Graphical User Interface

We have developed a graphical user interface (GUI) shown in Figure 7 that can be used by network administrators to specify a network by giving its routers and switches as well as the bandwidths of links, and monitor the allocation of various links. This GUI provides the following features:

1. The user may specify the backbone network nodes, namely routers and switches, and links using a drop-down menu. Our GUI currently supports the defining of ports as well as the connectivity between ports of different routers and switches as well as of ports within the same node. The bandwidth of each link also may be specified. Presently, we are creating a database of popular routers and their internal structures. This database will allow the user to select and configure routers from the defined set. Also, a user may add a new router or switch together with its hardware constraints to the database.

2. This system provides the time periods for which a particular link is already reserved by other requests.

3. The system can execute a scheduling algorithm, graphically display the computed path, and accept or reject this path as appropriate.

The design is hierarchical in nature and allows the user to navigate from aggregate to fine levels of information. The design has been demonstrated to be capable of representing complex networks and in particular the graph currently generated manually for USN.

4 UltraScience Net

UltraScience Net is a wide-area experimental network testbed to support the development of networking capabilities needed for next-generation computational science applications [7]. USN provides dedicated high-bandwidth channels for large data transfers, and also high-resolution, high-precision channels for fine control operations. The data plane of USN consists of four thousand miles of dual OC192 connections spanning Oak Ridge, Atlanta, Chicago, Seattle and Sunnyvale shown in Fig. 8. These connections are switched in the core

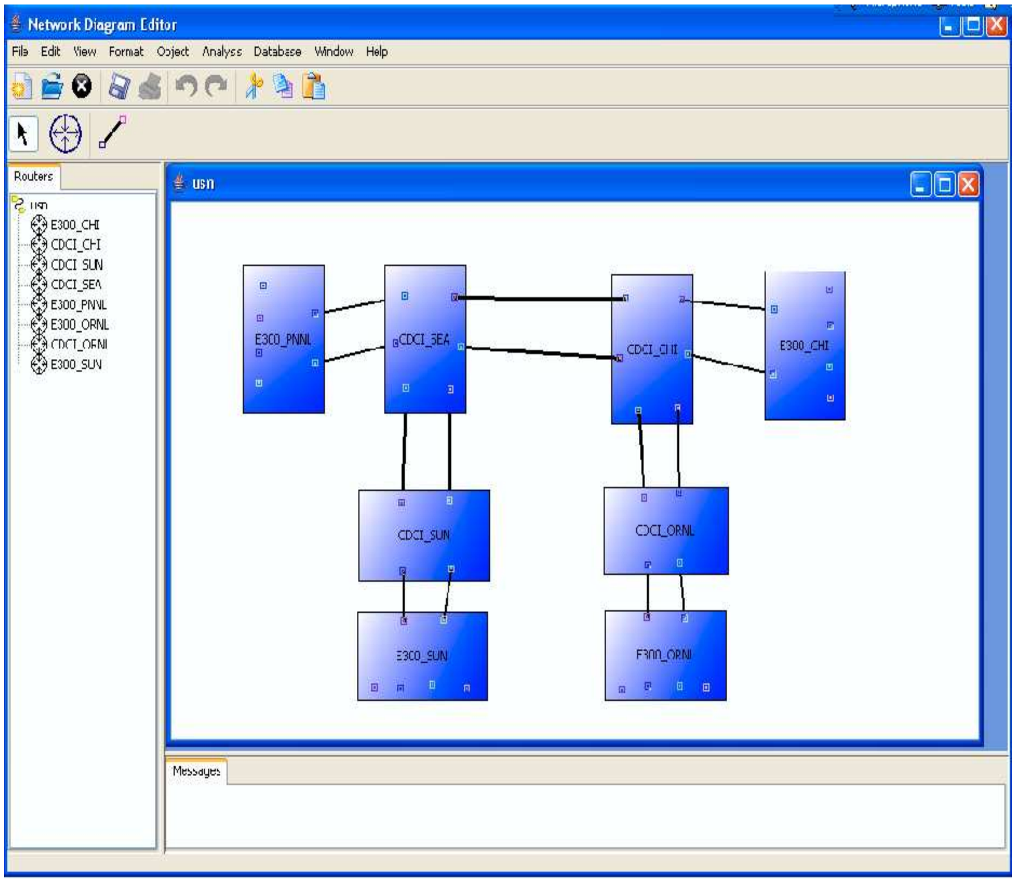


Figure 7: GUI for specifying network nodes and links.

using Ciena CD-CI switches and are provisioned at the edges using Force10 E300 switches. Dedicated channels of maximum 10Gbps capacity can be provisioned on USN at layer-1 and layer-2 at 150Mbps resolution. Layer-1 or SONET connections are provisioned exclusively between the CD-CI switches with capacities ranging from OC3 through OC192. Layer-2 channels are provisioned between E300 switches via SONET paths between core switches with capacities ranging from 150Mbps through 10Gbps. Also, layer-2 channels can be provisioned between core CD-CI switches but at lower capacities, ranging from 150Mbps through 1Gbps.

USN data plane can be configured into different channels that can separate the network in disconnected parts; thus by design, there is no expectation of data plane continuity, which in turn necessitated an out-of-band implementation of the control plane. Details of the

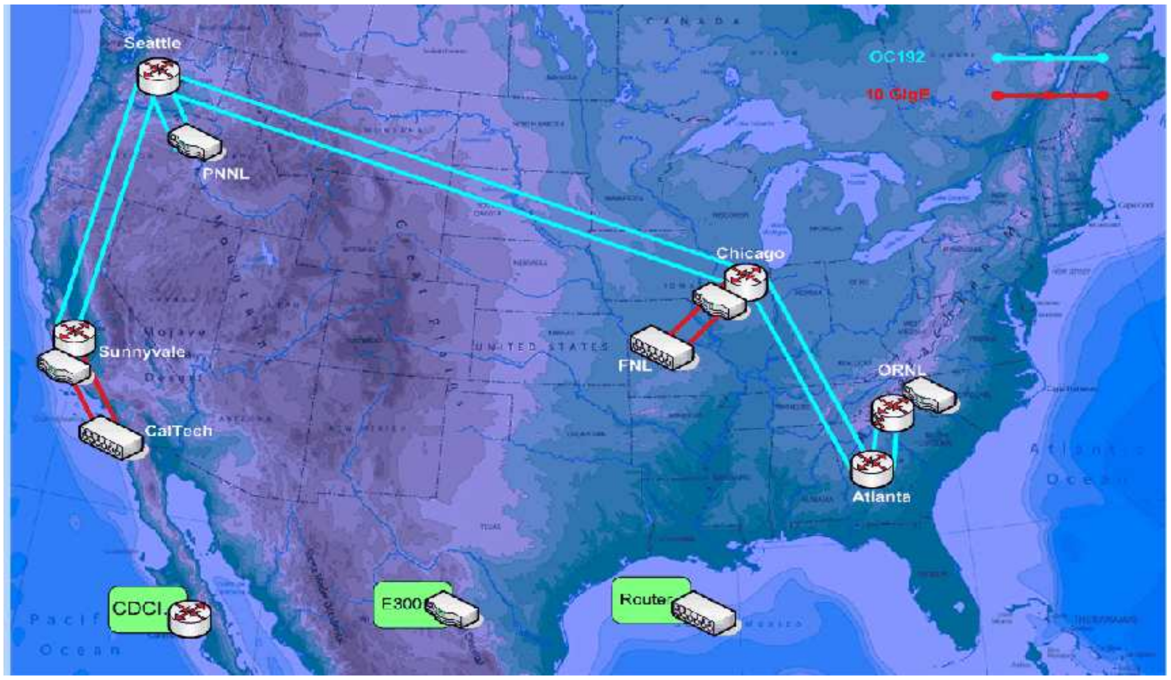
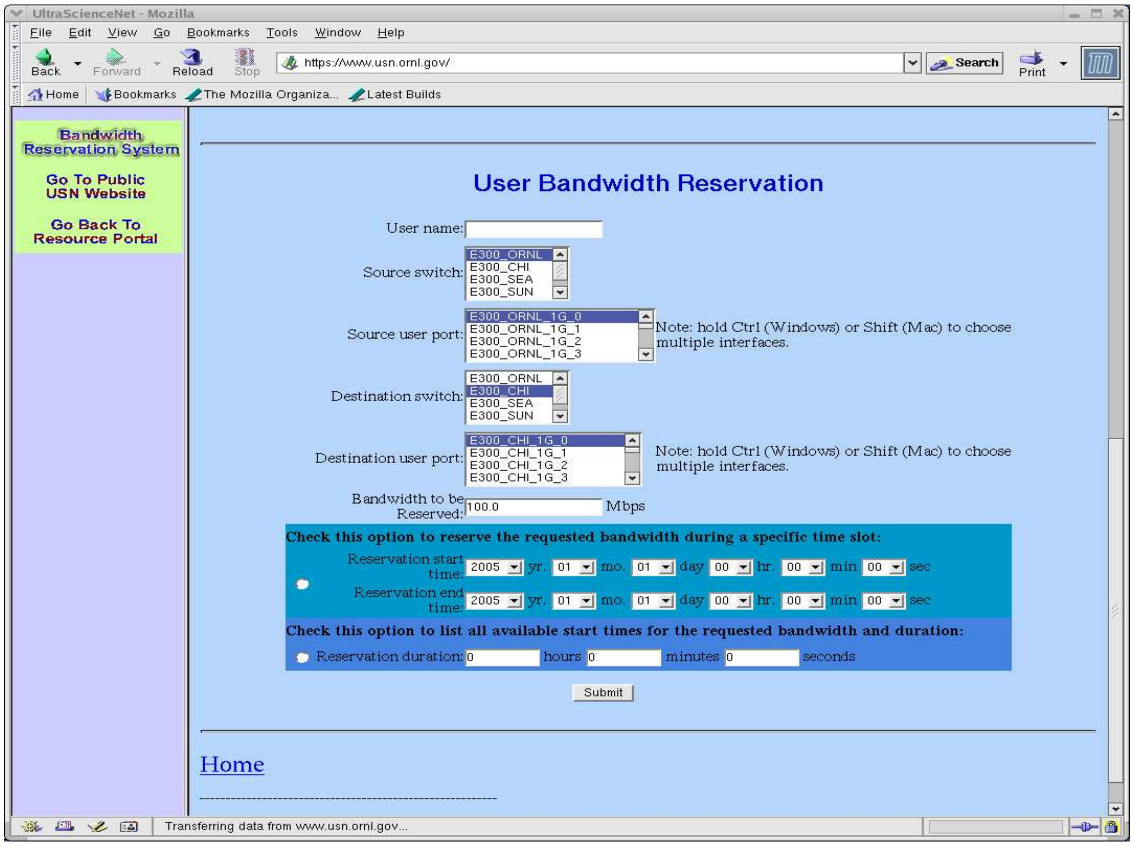


Figure 8: UltraScience Net data plane.

control-plane of USN can be found in [17].

User sites are connected to USN via their assigned ports on the edge or core switches. A user can request a dedicated path between two USN ports using the token-based authorization system. The VLAN (Virtual Local Area Network) tags are carried transparently across layer-2 USN channels, and hence VLANs, if employed, must be appropriately coordinated at both end points. After logging into the system with valid credentials, users can utilize a web-based graphical interface shown in Fig. 9 to make their bandwidth reservation requests and check the current allocations on various links. A web service is also developed to support web-clients based on WDSL (Web Service Description Language) description and SOAP-based communication of XML messages. An authentication system for the web clients as well as web users using X.509 certificates is also implemented.

The scheduling algorithms of Section 2 have been implemented and integrated into the USN control plane. The scheduled connections are set up and torn down using backend signaling. The CD-CI core switches are signaled using TL1 commands, and E300 edge switches are signaled using CLI commands. In both cases, EXPECT scripts are utilized by


UltraScienceNet - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
Search Print
Home Bookmarks The Mozilla Organiza... Latest Builds
Bandwidth Reservation System
Go To Public USN Website
Go Back To Resource Portal
Submit
Home

https://www.usn.ornl.gov/

User Bandwidth Reservation

User name:

Source switch: [E300_ORNL] E300_CHI E300_SEA E300_SUN

Source user port: [E300_ORNL_1G_0] E300_ORNL_1G_1 E300_ORNL_1G_2 E300_ORNL_1G_3 — Note: hold Ctrl (Windows) or Shift (Mac) to choose multiple interfaces.

Destination switch: E300_ORNL [E300_CHI] E300_SEA E300_SUN

Destination user port: [E300_CHI_1G_0] E300_CHI_1G_1 E300_CHI_1G_2 E300_CHI_1G_3 — Note: hold Ctrl (Windows) or Shift (Mac) to choose multiple interfaces.

Bandwidth to be Reserved: 100.0 Mbps

Check this option to reserve the requested bandwidth during a specific time slot:

Reservation start time: 2005 yr. 01 mo. 01 day 00 hr. 00 min 00 sec

Reservation end time: 2005 yr. 01 mo. 01 day 00 hr. 00 min 00 sec

Check this option to list all available start times for the requested bandwidth and duration:

Reservation duration: 0 hours 0 minutes 0 seconds

Transferring data from www.usn.ornl.gov...

Figure 9: Web interface of USN.

the signaling daemon to log in via encrypted VPN tunnels to issue the commands. The current version of USN control plane is implemented in C++ using CGI and PHP scripts for the server and JavaScript and HTML for user web interface. It is deployed on a central management node on a Linux workstation at Oak Ridge National Laboratory.

5 Conclusions

We presented bandwidth reservation and path computation algorithms that are applicable to connection-oriented networks. We have developed a GUI that allows automatic generation of graph models for these algorithms from user-specified network topology. We described a particular implementation of these algorithms that have been integrated into the USN control plane. Future work will focus on the development of algorithms for computing multiple paths, incorporation of latencies and other parameters as well as the distributed implementation of scheduling algorithms. In particular, it would be interesting to see if distributed versions of our algorithms can be integrated into "time-enhanced" versions of MPLS/GMPLS.

Acknowledgment

This research is sponsored by the High Performance Networking Program of the Office of Science, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC, National Science Foundation Under Grant No. ANI-0229969, an award from Advanced Algorithms and Systems, LLC and an award from the University of Florida.

References

[1] Dynamic resource allocation via GMPLS optical networks. http://dragon.maxgigapop.net.

[2] On-demand secure circuits and advance reservation system. http://www.es.net/oscars.

[3] N. Yamanaka and K. Shiomoto and E. Oki. *GMPLS Technologies*. CRC Taylor Francis Pub, 2006.

[4] P. Aukia and M. Kodialam and P. V. N. Koppol and T. V. Lakshman and H. Sarin and B. Suter. RATES: A server for MPLS traffic engineering. *IEEE Network*, pages 34–41, March/April 2000.

[5] X. Zheng and M. Veeraraghavan and N. S. V. Rao and Q. Wu and M. Zhu. CHEETAH: Circuit-switched high-speed end-to-end transport architecture testbed. *IEEE Communications Magazine*, 2005.

[6] N. S. Rao and S. M. Carter and Q. Wu and W. R. Wing and M. Zhu and A. Mezzacappa and M. Veeraraghavan and J. M. Blondin. Networking for large-scale science: Infrastructure, provisioning, transport and application mapping. In *Proceedings of SciDAC Meeting*, 2005.

[7] N. S. V. Rao and W. R. Wing and and S. M. Carter and Q. Wu. Ultrascience net: Network testbed for large-scale science applications. *IEEE Communications Magazine*, 2005. in press, expanded version available at www.csm.ornl.gov/ultranet.

[8] Z. L. Zhang and Z. Duan and Y. T. Hou. Decoupling QoS control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *Proc. ACM SIGCOMM*. 2000.

[9] Z. L. Zhang and Z. Duan and Y. T. Hou. On scalable design of bandwidth brokers. *IEICE Transactions on Communications*, E84-B(8), 2001.

[10] U. Black. *MPLS and Label Switching Networks*. Prentice-Hall Pub., 2002.

[11] Enlightened Computing, http://www.enlightenedcomputing.org/.

[12] Geant2, http://www.geant2.net.

[13] Hybrid Optical and Packet Infrastructure, http://networks.internet2.edu/hopi.

[14] Internet2. http://www.internet2.edu.

[15] JGN II: Advanced Network Testbed for Research and Development, http://www.jgn.nict.go.jp.

[16] LHCNet: Transatlantic Networking for the LHC and the U.S. HEP Community, http://lhcnet.caltech.edu/.

[17] D. Ghosal and B. Mukherjee N. S. V. Rao and Q. Wu and S. M. Carter and W. R. Wing and A. Banerjee. ontrol plane for advance bandwidth scheduling in ultra high-speed networks. In *INFOCOM 2006 Workshop on Terabits Networks*, 2006.

[18] S. Sahni. *Data structures, algorithms, and applications in C++*. Silicon Press, 2005. Second Edition.

[19] User Controlled LightPath Provisioning, http://phi.badlab.crc.ca/uclp.