NIST Switch: A Platform for Research on Quality of Service Routing

Mark Carson[a] and Michael Zink[b]

[a]National Institute of Standards and Technology (NIST),
University of Maryland, College Park

[b]National Institute of Standards and Technology (NIST),
University of Technology, Darmstadt (Germany)

ABSTRACT

We have designed and are implementing a research platform, NIST Switch, for experimenting with novel approaches to routing with quality of service (QoS) guarantees in an IP environment. NIST Switch is based on commodity PC hardware running freely-available operating systems (FreeBSD and Linux). It implements quality of service through label switching over Ethernet (10Mb and 100Mb), using proposed extensions to RSVP and differentiated services to signal QoS requests and distribute labels.

In NIST Switch, the labels applied to packets have two fundamental uses in QoS traffic management: Locally, labels select the characteristics of the queueing and traffic shaping measures applied to packets. In the network, labels identify the path segments ("sticks" or "brushes") over which traffic will flow. A label-switched route (for either unicast or multicast traffic) then consists of a "bundle" (path-connected subset) of these sticks and brushes, chosen so as to achieve the appropriate aggregate behavior. NIST Switch routing algorithms attempt to optimize the set of path segments used, so as to maximize the number of service requests which can be met while bounding the number of labels allocated.

As an experimental platform, NIST Switch is designed to be easily altered. Each of its key components (queueing algorithms, label databases and distribution, routing algorithms, device support routines, QoS policies) are independently configurable modules which are readily replaceable.

Keywords: label switching, quality of service routing, MPLS, RSVP, CBQ

1. INTRODUCTION

NIST Switch is a platform for experimenting with *quality of service routing* based on *multiprotocol label switching*. We briefly discuss both concepts, stressing those aspects which are useful from our viewpoint.

Quality of service (QoS) in this context means the ability to guarantee some minimum handling requirements for a communications stream. Typical QoS requirements include bandwidth guarantees (e.g., at least 2 Mbits/second) or latency guarantees (e.g., no more than 100ms end-to-end transit time). QoS requirements may be probabilistic rather than fixed (e.g., 2 Mbits/second available at least 90% of the time), or may be relative rather than absolute (e.g., preferred traffic may be less likely to be dropped, or may face shorter average queue times, when a router is overloaded). QoS routing attempts to determine and allocate routes through a network which meet these QoS guarantees.

Multiprotocol or *multilayer label switching* is, in a broad sense, the use of layer 2 (in the OSI model) constructs — "labels" or (in Cisco's original terminology) "tags" — such as (possibly extended) Ethernet headers or Asynchronous Transfer Mode (ATM) virtual circuit (VC) or virtual path (VP) identifiers, for the higher-level purposes of routing and QoS provisioning.

Label switching is referred to as *multiprotocol* since it is envisioned as a way of accommodating the "advanced" routing needs of multiple higher-level protocol types with the facilities of a particular link layer (in particular, ATM running both IP and native protocols).

Send correspondence to mark.carson@nist.gov.

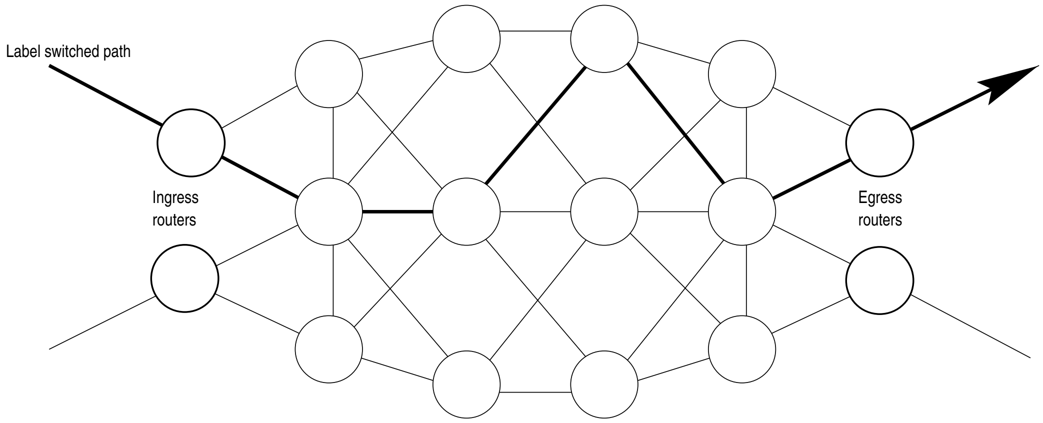

Figure 1. Label-switched path within a domain

It is also called *multilayer* because in effect we are encoding routing and session information, normally considered as being part of layer 3 and higher in the OSI model, into layer 2 labels. Label switching is thus sometimes referred to as "layer $2\frac{1}{2}$ routing" since, while it is based directly on information stored in layer 2 headers and is generally restricted to layer 2 domains, it is intended to address higher-level concerns.

Labels have meaning within some area, called a *domain*. A domain is assumed to have central administration, so that labels will have a uniform agreed-on meaning within the domain. Domains are in effect layer $2\frac{1}{2}$ groupings, possibly larger than ordinary subnets, but not encompassing the entire network. Within the domain, labels generally prescribe a path segment or subtree, referred to as a *label-switched path* (see figure 1).

Routers within a domain may be internal only (with no connections to routers outside the domain), or may be *border* routers, with external connections. In some cases, we differentiate between *ingress* and *egress* routers, though of course normally a border router will fulfill both functions at various times.

In general, a packet may have multiple labels — a *label stack* — assigned to it (see figure 2). The topmost label on the stack is the one actually used, until the intermediate destination it indicates is reached. At this point, the top label is popped from the stack, and the next label, if any, is used. Typically, multiple labels are used when packets are directed for part of their trip through some subdomain which may use different link technology, or may aggregate traffic in some way. In this case, a subdomain label is pushed on top of the stack for use within the subdomain; on exit, the previous label is used. Label stacks may also be useful for other purposes; here, multiple labels are used within a single domain to create aggregates (bundles) of label-switched paths for routing.

The original impetus behind label switching was to make more effective use of the QoS and routing facilities provided by ATM. Direct assignment of ATM VP/VC ids based on knowledge of the needs of IP routing allowed for intelligent implementations of such features as IP multicast. These were much more efficient than the naive methods (such as routing over VC meshes) in previous use. The possibilities this illustrated has since led to the extension of label switching to other link layer technologies, even ones such as Ethernet with few native QoS or routing facilities to exploit.

Label switching is thus interesting from a research standpoint because it provides a new networking construct with as yet no firmly fixed associations. Labels are potentially useful for many purposes: bandwidth or latency guarantees, service differentiation, traffic management, multicast group definition, load balancing, more efficient routing.

The fact that labels have significance only within a domain, and not end-to-end, is actually an advantage from this viewpoint, since it allows for independent label-based experiments with no need for large-scale *a priori* agreements. With labels normally set within an intermediate domain, there is no reliance on them being set properly by a naive originating host, assuming there is some other means within the experimental environment of determining the proper treatment of the traffic.

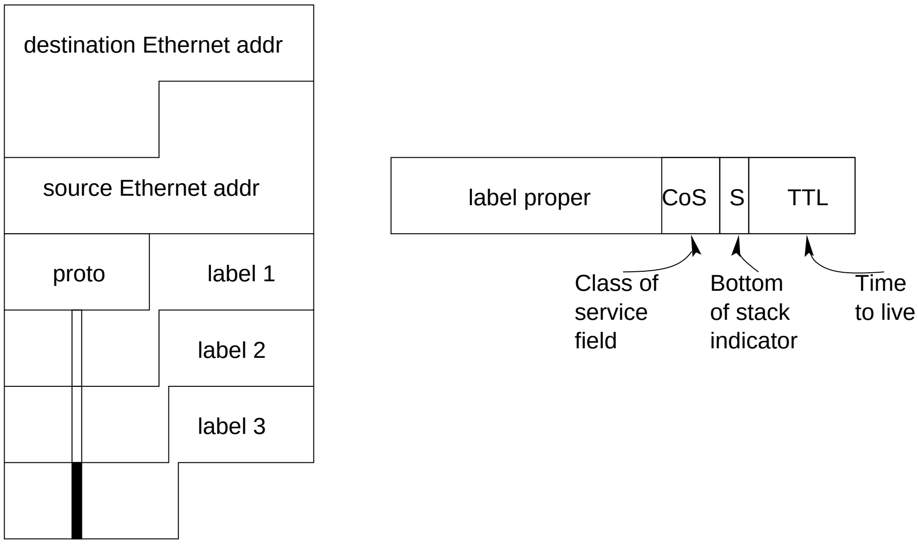

Figure 2. Ethernet label stack and label structure ("Shim" version)

Ultimately, as certain uses for labels prove successful, they may prove of interest between domains or even end-to-end; for this reason, label interchange protocols, or interoperation with higher-level end-to-end protocols are also of interest.

As the previous remarks were intended to indicate, label switching is a technique which offers much potential for improved performance and capability in routing over a variety of link layer types. However, taking best advantage of this potential can be quite complex, requiring some period of experimentation and development. Unfortunately, there are no generally available non-proprietary platforms to allow interested outside parties to experiment with label switching technology. This limits the potential community which can contribute to the ongoing research and development in this area. It is for this reason that we have undertaken the development of an inexpensive commodity-hardware-based research environment, NIST Switch, which provides the necessary facilities for label switching experimentation. The base system requirements are deliberately minimal, consisting of only ordinary PC hardware with common Ethernet cards (and/or, later, ATM interfaces).

In this paper, we briefly describe the NIST Switch architecture, then discuss some current experiments we are conducting with NIST Switch in support of end-to-end QoS signalling protocols, and domain-wide QoS routing.

2. NIST SWITCH ARCHITECTURE

For a project of this sort, the accessibility of its software is as important as that of its hardware. NIST Switch is based on freely-available, widely-deployed operating systems (Linux and FreeBSD). Where possible, the individual components of NIST Switch itself are based on freely-available building blocks, such as ALTQ from Sony,[8] and the GateD OSPF implementation.[15] To facilitate experimentation, all the essential parts of the system (routing tables and algorithms, queueing algorithms, label assignment and distribution, network device handling) are set up to be easily modified and replaced, even during runtime.

The essential components of NIST Switch operate at a new "layer $2\frac{1}{2}$" protocol handler in the kernel, which sits between the Ethernet link layer protocol handler routines and the IP protocol handler code. The intent is to provide as clean a separation of the label switching code as possible, though some small modifications to the generic Ethernet and IP code are required. In figure 3 are shown the principal components of NIST Switch and their interdependencies.

We briefly summarize all the architectural components, then go into more detail on a few areas of active research.

1. Basic label handling module:

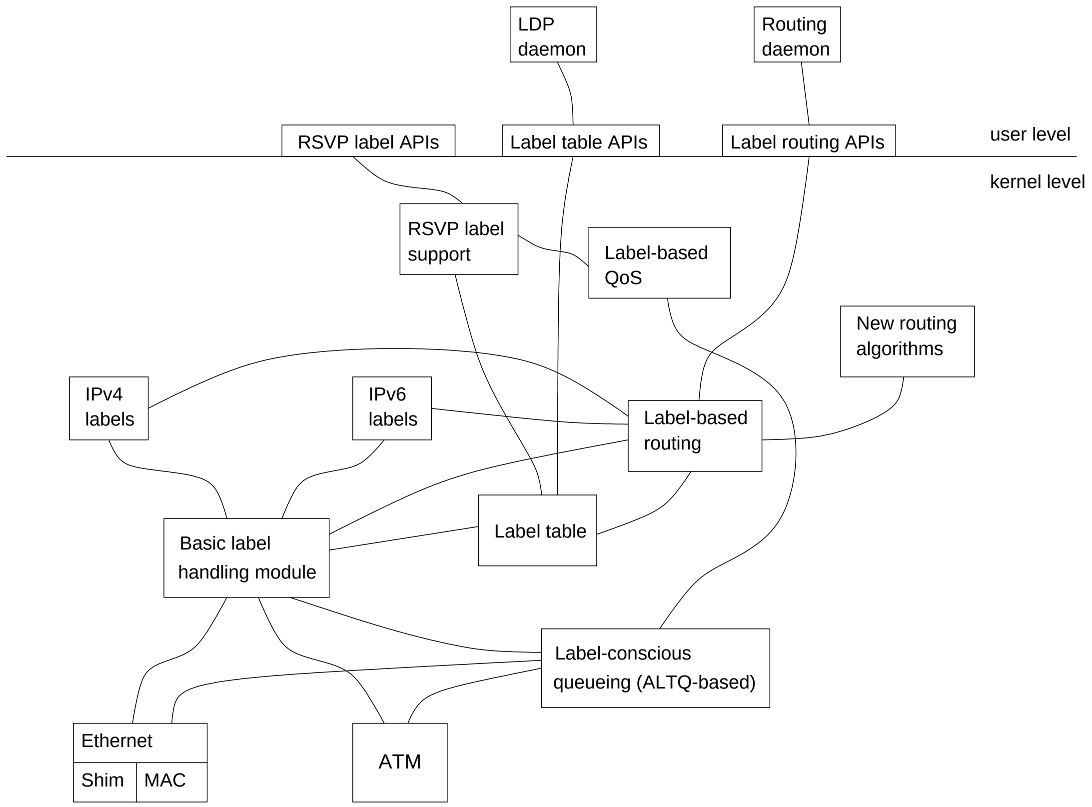

Figure 3. NIST Switch Architecture

Creates, interprets and processes the labels (or label stacks) for all incoming and outgoing packets. For outgoing packets, it determines the appropriate label (if any) to apply and the output interface for the packet based on the source and destination addresses (IP address and possibly UDP/TCP port), route (for explicitly routed packets), and class of service. For incoming packets, it uses the packet's label to determine how the packet should be handled (forwarded or received locally). If the incoming packet is to be forwarded, it determines what changes (if any) should be made to the packet's label stack. The current label may be popped from the stack or replaced by a new label, or a new label may be pushed on top of the stack.

2. Label table:

   Label information is kept in a kernel table, indexed by both label and by higher-level source, destination, route, and service information. The table is designed to be accessible by multiple simultaneous readers (network interrupt-level packet forwarding routines) and a single writer (task-time label table update routines, running on behalf of RSVP[5] or simple LDP label distribution protocols,[1] or QoS routing daemons). The table is assumed to be potentially fairly large, with perhaps tens of thousands of entries for a domain of hundreds of nodes. Fast indexing is thus required; here, it is provided by a variant of dynamic double hashing.[10] While indexing by label (for example, when forwarding an already labeled packet) requires only a single lookup, indexing by higher-level information (as is performed by ingress routers) may require multiple lookups to find the best matching entry.

3. Link level label handling:

   The link level label handling code sets and retrieves the label information in the (extended) Ethernet (or, at a future stage, ATM) packet header. For Ethernet, both "MAC"[6] and "shim"[13] versions of Ethernet labels are supported. Either type is recognized on input from any interface; the type used (if any) on output is configurable on a per-interface basis.

For simplicity, we do not currently fragment packets which have grown too large by the addition of label stacks for the 1514-byte Ethernet limit, but instead rely on the ability of recent Ethernet cards and switches to accommodate oversized packets. Later, we will implement fragmentation/packet dropping, as required by the interface MTU.

4. Label-conscious queueing

   To provide support for quality of service guarantees, we need queueing algorithms which classify and handle packets appropriately. Code for these exists for pure IP implementations (ALTQ for FreeBSD, new 2.1.1XX code for Linux). We adapted the CBQ routines from these implementations to include support for a label-based environment. The adapted queueing code handles all traffic, both labeled and standard IP packets, in a unified way with a single set of queues, based on the service class the packets belong to.

5. RSVP label support

   RSVP interacts with labels in two basic ways. As a QoS signalling protocol, it distributes requests for the allocation of resources for particular flows, which we typically associate with labels. Also, RSVP can serve as a distribution protocol for labels.[7] We implement both types of RSVP label support. (RSVP support is currently limited to the FreeBSD version, pending the release of a compatible RSVP implementation for Linux 2.1.1XX.) The modifications to RSVP required for label support are discussed in 3.

6. Differentiated services label support

   Diffentiated services[3] can be viewed as an attempt to provide fairly direct input into traffic management facilities. The interactions between differentiated services-style QoS and labels is an area of active discussion at the Internet Engineering Task Force (IETF). Pending further work in this area, we at this point simply use the differentiated services byte as part of the key in determining the packet label, assuming the label table entries have been set properly beforehand.

7. Labels and routing

   One of our prime research interests is the use of label-switched paths as units for more efficient QoS routing. This is discussed in 4.

8. User/daemon label support

   We provide a set of user-level APIs which allow reading and updating the kernel label database. These will used by the LDP daemon to get, distribute, and update label information.

9. Label distribution

   We will eventually support two methods of label distribution: currently, through the RSVP *label-object*, and in the future through a simple user-level LDP daemon. The LDP daemon communicates with its peers through (a subset of) the proposed IETF LDP protocol.[1] The LDP daemon will be used to distribute information for differentiated services-related label entries.

10. Label routing daemon

    The label routing daemon is responsible for distributing routing information and the associated labels within the domain, and for appropriately advertising (normally) non-labeled versions of this information outside the domain. Within the domain, we have devised a simple hack to the OSPF implementation in GateD.[15,14] Routers advertise label capability through an option bit in the link state header. When two label-capable routers communicate, they interchange label information in an extended router link state record.

    Between two labeled domains, there is current work in the IETF on integrating label information into a modified BGP routing protocol.[12]

3. LABEL SWITCHING AND RSVP

For the initial version of NIST Switch, we have implemented label distribution through an extended version of RSVP.[7] This approach allowed us to reuse existing code to handle the details of distribution of label information on the appropriate paths. This has also the advantage that the QoS signalling, the resource reservation and the label

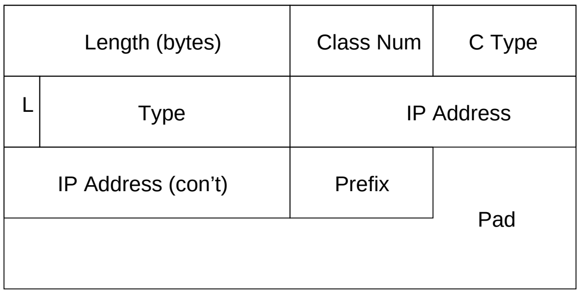

Figure 4. Explicit Routing Object (IPv4)

handling for flows are all combined in one protocol. We also extended RSVP with the addition of explicit routing.[9] This makes it possible to determine on which path RSVP messages, including our label informaton, travel through a network. This allows us to do various sorts of traffic engineering.

In a third step we modified the ALTQ module[8] in a manner that allows it to do class-based queueing on the basis of labels instead of IP headers.

3.1. Explicit Routing in RSVP

The proposed extension of explicit routing[9] makes it possible to send RSVP messages on a different path than it would be done by standard IP routing. This is useful in the case where a network operator wants to direct QoS traffic on certain routes through the network. In our case it has the additional benefit that we can dictate where in the network label-switched paths should be set up, and can ensure through appropriate route allocation that we avoid anomalies such as routing loops. Explicit routing in RSVP in combination with label switching also has the advantage that no changes have to be done on the standard IP routing, as is discussed below.

In order to be able to do explicit routing in RSVP, we extend it with the explicit routing object shown in figure 4. This object must only be included in PATH messages, since RESV messages always travel the same path in opposite direction. Thus, no change for the handling of RESV messages is necessary. When an explicit routing object is present in a PATH message, the RSVP interface to the standard routing is bypassed and the path message is sent out the interface that leads to the specified IP address. We extended the RAPI[4] to allow setting up an explicit path, by determing the hosts (strict routing) or the networks (loose routing) the PATH message should be sent to.

3.2. Label distribution

As described in Two new RSVP objects are required for label support in RSVP, the *label-request-object* and the *label-object*.[2] The *label-request-object* is included in the PATH messages and signals that a label should be allocated for this request. In the unicast case, the label is allocated at the downstream destination hop, and inserted in a *label-object* which is then transported back upstream via the RESV message. (Label allocation at egress.) This label is then used in the upstream host as the outgoing label in the label tabel (Chapter 2). Figure 5 shows in a very simple example how label distribution via RSVP in the unicast case is done.

Since we assume that label switching, at least in the near future, will only be done at intermediate domains, within the core of the network, we have our ingress routers automatically insert a *label-request-object* into the PATH messages. For the unicast case, the *label-object* is allocated at the egress router. All label-related objects are removed at any egress from the domain so they will not be propagated to non-label-aware endsystems.

In order to avoid multiple use of the same label within a domain, we reserve a range of labels for each interface of each label-capable system. Currently, this range is simply set up manually. In the future, this will be determined dynamically by a label distribution server which allocates subsets of labels to the switches as needed.

In the case of RSVP multicast messages, label distribution is performed in a different fashion from the unicast case (see figure 6). If a border system receives a RSVP PATH message with a multicast destination address, it puts a *label-object* in addition to the *label-request-object* in the PATH message. (Label allocation at ingress.) The same label is distributed to all MPLS routers in the multicast group. This label is then returned in the RESV messages

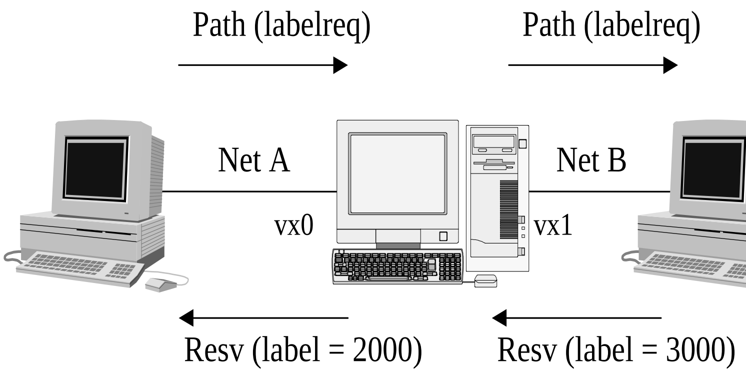

Figure 5. Label Distribution (Unicast)

transported back upstream. This is from our point of view a much more convenient way than described elsewhere,[7] because of the fact that no label renegotiation is required.

4. LABEL SWITCHING AND ROUTING

A key idea in label switching is that the use of labels can effect some centralization in route determination. Unlike IP-only routing algorithms, which can only make single-hop determinations, a label switched environment can create multiple-hop path segments or tree structures by distributing appropriate label information. Typically, labels are generated for a set of covering paths or trees for some network, and incoming packets are labeled for the appropriate path or tree. The path determination then needs only to be done once or some limited number of times; at the intermediate nodes, the label alone determines the next hop.

This has a number of advantages:

- Single-hop route determinations are subject to various anomalies such as routing loops which can be difficult to detect and fix. With labels, we can generate and maintain individual routes and route segments as desired, forestalling the development of any loops or transient misdirection of packets.
- When separate routing decisions are being made at every node, it is very difficult to ensure that end-to-end quality of service constraints are met. When labels are used, we can, through control of label generation and assignments, coordinate the necessary resource reservations and queue prioritizations to guarantee bandwidth levels and control overall packet delay times.
- Also, since routing decisions can be made at fewer points, and even made a priori (in whole or in part), there is more room for implementing more complex routing algorithms which may be capable of providing better path determinations and better overall network utilization. In particular, the idea favored here of routing via a collection of moderate-length path segments or trees is an area as yet relatively unexplored even theoretically; until now, most routing algorithms have assumed either single-hop or end-to-end decision-making.

Our basic approach to label-mediated QoS routing is in certain ways similar to the segmented routing used in routing FPGA designs.[11] It may be summarized as follows: In general, a label designates a (label-switched) path or tree segment ("stick") in a network with (possibly) certain QoS characteristics. These "sticks" may be preallocated or (particularly for RSVP-created paths) made as needed. The set of sticks will in general form a covering forest for the network graph.

Routing at the entry gateway then consists of selecting the appropriate "bundle" of one or more of these sticks, and applying the corresponding label stack to the packets. Then, through the rest of the net, the labels dictate both the route taken and the traffic handling characteristics applied. Desirable routes are those with a small bundle size, and hence small label stack (preferably one only).

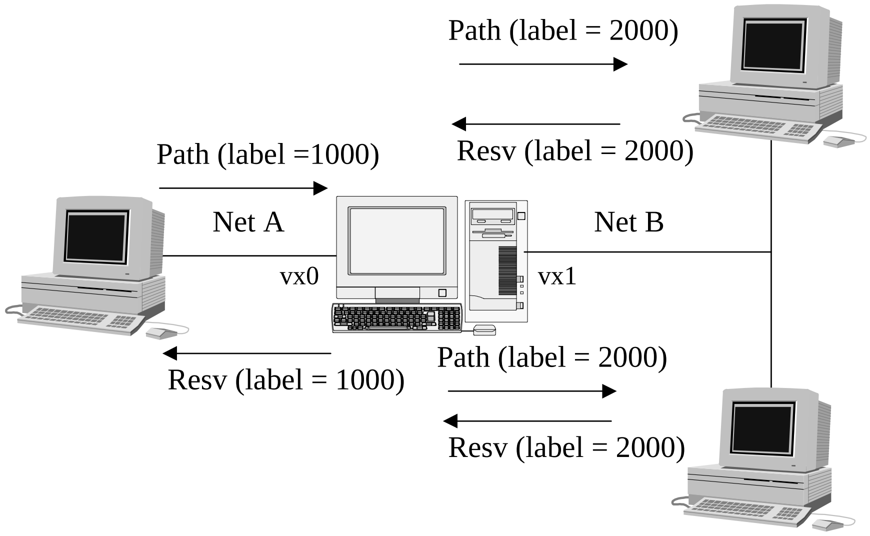

Figure 6. Label Distribution (Multicast)

Determining good routes requires two steps: the *a priori* allocation of useful sticks, and the on-demand concatenation of these into appropriate bundles. From a graphic-theoric viewpoint, the questions of interest then become how to allocate a labeled covering forest and how to manage it to achieve the desired routing goals. Typically the number of labels allowed will be fairly limited (say, in the low thousands), allowing only some fraction of the set of possible useful paths to be specified. For both of these, we assume some knowledge of the state of the entire domain.

At each potential ingress router, we store information on the structure of the entire domain, including the interconnections of the nodes and the type of connection (here, 10Mb or 100Mb Ethernet). We consider this as a graph with lengths (latencies) and capacities (bandwidths) on the edges. Currently, we simply set all edge lengths to 1 and all capacities to 10 or 100. From this information, we run a heuristic for the allocation of good sticks. Our current best heuristic is a sort of randomized greedy length-bounded augmenting path algorithm. Here is a brief description:

- First, reserve some fraction (for the sake of simplicity, 1/2) of the label table space for "on-demand" allocated labels (through RSVP or the like).
- Create a larger graph G' by the addition of two nodes s and t. Connect all the sources (ingress routers) to s and all the destinations (egress routers) to t by links of zero length and infinite capacity. This is merely a technical point to run the flow algorithm below. Since the sets of sources and destinations will overlap, and in fact are in general the same set, this will create many two hop paths from s to t; we ignore these below.
- For the remaining pool of labels to be preallocated, proceed as follows until space is exhausted:
  - Choose a length bound $l \leq N$, where N is the number of nodes. (We run this repeatedly for an increasing sequence of l values up to N.)
  - Find a path from s to t of length at most l of maximum residual capacity.

    We find this recursively by randomly choosing an outgoing edge of residual capacity at least that of the incoming path if possible; where this is not possible, we choose the outgoing edge of maximum residual capacity. If this path construction has not reached t within l steps, we drop this path and start again.
  - Reserve the flow associated with this path in the graph G'. A variant procedure is to allocate only a fixed fraction (say, 1/2) of the available flow. This greatly increases the running time (by a factor of 4 or more), but does result in a better spread of useful routes.

- While in practice, this procedure produces good labeled segments between most nodes, some unevenness of flow distribution is possible. We are currently working on useful improvement procedures; in the interim, we simply free some flow between overly-well connected nodes, and rerun the algorithm on the subset of less-well connected nodes.

To allocate bundles, we create a multigraph of all the allocated sticks. In this multigraph, we then look for a shortest path from the source to the destination, ignoring any edges which do not have the requested capacity available. If allocating this bundle involves a latency or bandwidth guarantee, we adjust the multigraph as follows: For a latency guarantee, we increase the recorded latency of the sticks used by 1. For a bandwidth guarantee, we decrease the recorded bandwidth by the amount guaranteed. In practice, even with graphs of thousands of nodes, the typical bundle size is no more than 3 or 4, so this algorithm runs quite quickly.

5. CONCLUSION

Label switching provides a useful hook for many areas of research interest in routing and quality of service provisioning. The NIST Switch architecture is intended to facilitate such research efforts by providing a low-cost, easily accessible development environment.

For more information and (eventual) source code access, the NIST Switch web site is *http://www.antd.nist.gov/itg/nistswitch.*

REFERENCES

1. L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas *LDP Specification, (draft-ietf-mpls-ldp-01.txt).* INTERNET DRAFT, Internet Engineering Task Force, August 1998.
2. D. Awduche, D. Gan, T Li, G. Swallow, V. Srinivasan. *Extensions to RSVP for Traffic Engineering, (draft-swallow-mpls-rsvp-trafeng-00.txt).* INTERNET DRAFT, Internet Engineering Task Force, August 1998.
3. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. *An Architecture for Differentiated Services (draft-ietf-diffserv-arch-01.txt).* INTERNET DRAFT, Internet Engineering Task Force, August 1998.
4. R. Braden, D. Hoffmann. *RAPI – An RSVP Application Programming Interface Version 5, (draft-ietf-rsvp-rapi-01.txt).* INTERNET DRAFT, Internet Engineering Task Force, August 1998.
5. R. Braden, et al. *Resource ReSerVation Protocol (RSVP) – Version 1 Funcitional Specification.* RFC 2205, Internet Engineering Task Force, September 1997.
6. D. Bussiere, H. Esaki, A. Ghanwani, S. Matsuzawa, J. Pace, V. Srinivasan *Labels for MPLS over LAN Media, (draft-srinivasan-mpls-lans-label-00.txt).* INTERNET DRAFT, Internet Engineering Task Force, August 1997.
7. B. Davie, Y. Rekhter, E. Rosen, A. Viswanathan, V. Srinivasan, S. Blake. *Use of Label Switching with RSVP, (draft-ietf-mpls-rsvp-00.txt).* INTERNET DRAFT, Internet Engineering Task Force, March 1998.
8. K. Cho. *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers.* In Proceedings of USENIX 1998 Annual Technical Conference, New Orleans LA, June 1998.
9. D.-H. Gan, R. Guerin, S. Kamat. *Setting up Rerservations on Explicit Paths using RSVP, (draft-guerin-expl-path-rsvp-01.txt).* INTERNET DRAFT, Internet Engineering Task Force, November 1997.
10. D. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Springer-Verlag, 1984.
11. G. Lemieux, S. Brown, D. Vranesic "On Two-Step Routing for FPGAs,," *Proc. ISPD-97*, ACM, April, 1997. See also W. Moore, W. Luk, ed. *Oxford 1991 International Workshop on Field Programmable Logic and Applications.*
12. Y.Rekhter, E. Rosen. *Carrying Label Information in BGP-4, (draft-ietf-mpls-bgp4-mpls-01.txt).* INTERNET DRAFT, Internet Engineering Task Force, August 1998.
13. E. Rosen, Y. Rekhter, D. Tappan, D. Farinacci, G. Fedorko T. Li, A. Conta. *MPLS Label Stack Encoding on LAN Media, (draft-rosen-mpls-lan-encaps-00.txt).* INTERNET DRAFT, Internet Engineering Task Force, November 1997.
14. J. Moy. *OSPF Version 2.* RFC 2328, Internet Engineering Task Force, April 1998.
15. See *http://www.merit.edu/ gated/.*