Simple Active Mechanisms for Measuring and Monitoring Service Level Topologies

Gísli Hjálmtýsson, Ólafur Ragnar Helgason, and Björn Brynjúlfsson*

Network Systems and Services Laboratory
Department of Computer Science
Reykjavik University, Reykjavik, Iceland
{gisli,bjorninn,olafurr}@ru.is
http://netlab.ru.is

Abstract. Whether driven by security concerns, need for flexibility, deployment of advanced services or as a simplified outsourcing model, overlaying a virtual service topology over the underlying network infrastructure is common. To ensure and enforce consistent service quality, fairness and protocol behavior it is necessary to measure and monitor these service level topologies. In this paper we present extensible general purpose mechanisms to monitor and measure characteristics of a service level topology at the nodes of the topology. The mechanisms provide means to dynamically deploy a distributed observation function at the nodes of the topology and to collate the observations into a result given to the requestor on a subscription channel. These are control plane mechanisms, outside of the router datapath, where we assume programmability and low cost memory. We give several examples of how to use these mechanisms to compute interesting properties of the topology.

Keywords: Service aware networking, measurement, service management, overlays, VPN.

1 Introduction

Whether driven by security concerns, need for flexibility, deployment of advanced services or as a simplified outsourcing model, overlaying a virtual service specific topology over the underlying network infrastructure is a common solution approach for all types of network technologies. Static service topologies include "private networks" built using leased telephony lines to ensure isolation, and Internet "bones" based on statically allocated IP-in-IP tunnels such as the MBONE, 6-BONE and the A-bone, to provide the illusion of universal deployment over a virtual topology. More dynamic approaches include Virtual Private Network (VPN) services in ATM [1] established from dynamically allocated collection of ATM circuits, and Service Level Routing (SLR) over the Internet providing non-local redundancy and load balancing across a network of service locations [2]. Similarly, application level services, such as application layer multicast, build a service specific topology. Many proposed

* This work (all authors) was supported in part by The Icelandic Centre for Research under grant number 020500002.

G.J. Minden et al. (Eds.): IWAN 2004, LNCS 3912, pp. 205–216, 2007.

active networking services explicitly or effectively build an overlay topology. Recent interest in overlay networks can be viewed as an attempt to abstract out both the underlying topology and specific network technology.

To ensure and enforce consistent service quality, fairness and protocol behavior it is necessary to measure and monitor these service level topologies. The detailed technologies used to realize the various virtual topologies differ. However, significant and important commonalities make it desirable to design and implement general purpose mechanisms to support the monitoring and measurements and algorithms to compute values of common interest, rather than have each technology, service or application implement a fraction of such mechanisms.

End-to-end measurements and edge based solutions [3,4,5,6], requiring no cooperation from the core network, have been proposed to infer the service topology for multicast and network characteristics from edge observations. A significant drawback of these methods is that they only compute long term averages are inherently error prone and do not adapt well to membership changes in the service topology. We believe that rather than employing service ignorant long term observations from the edges, effective management of service level virtual topologies requires service specific observations from inside the service network for timeliness and relevance.

Commonly, virtual topologies are built from nodes such as boundary nodes where functionality beyond basic forwarding is common. These nodes are prime candidates and targets of active networking technology. Moreover services based on P2P networks, application layer multicast and overlays that are implemented at the application layer can reasonably be assumed to have high level of activity and programmability. Adding mechanisms for observations at the nodes of the virtual topology is therefore achievable.

Such general mechanisms must provide a) the right abstractions for results of importance, b) extensibility to support service specific observations, and c) interface that makes it easy to employ with a variety of service level topology approaches, such as application level overlays, ATM style VPN's, IP-bones and network layer multicast.

In this paper we present extensible general purpose mechanisms to monitor and measure characteristics of a service level topology at the nodes of the topology. These are control plane mechanisms, outside of the router datapath, where we contend assumptions of high programmability and low cost memory are valid. Moreover, we present algorithms to collect and use these observations to compute interesting properties of the topology. The mechanisms provide means to dynamically deploy a distributed observation function at the nodes of the topology and to collate the observations into a result given to the requestor on a subscription channel. We give examples of our use of these mechanisms in managing multicast distribution, as well as in an control overlay for router selection in sparsely deployed services. We furthermore discuss the use of our mechanism to collect information for load distribution in network wide service level routing.

The rest of the paper is organized as follows. In Section 2 we discuss related work and further position our work. In Section 3 we describe the basic mechanisms, and show in Section 4 how we have used them to monitor and manage multicast service over a dynamically constructed virtual service topology. In Section 5 we discuss how the same mechanisms are used in general overlay/virtual topologies with two examples of control level overlays for service aware route selection. Section 6 contains additional discussion. In Section 7 we conclude.

2 Related Work

Network tomography from end-to-end measurements has been proposed to infer service level topology and network characteristics from edge observations [3,4,5,6]. An attractive attribute of these methods is that they don't require any cooperation from the core network and therefore work for large scale discovery across multiple administrative domains. However, a major drawback of these methods is that they only compute long term averages and are therefore inherently error prone and do not adapt well to membership changes. While valuable as a fallback, effective service management requires more detailed and timely observations that can only be obtained inside the service network.

Overlays have been used for monitoring the underlying physical network for path outages and periods of reduced performance. In [7], path restoration in the overlay network is done by finding a route for the backup path that minimize the probability that the primary and backup overlay paths share a link in the underlying network. In [8] aggressive probing between application layer overlay nodes is used to do fault detection of Internet paths and recovery is performed by routing by way of the overlay nodes instead of the IP routing. Although using overlays to monitor the underlying physical network can be a powerful approach, with our mechanisms we are primarily interested in monitoring the overlay and service level topologies themselves. We are not aware of any mechanisms for monitoring general overlay and service level topologies.

Related to the monitoring of general service level topologies is the monitoring of multicast. A number of mechanism and tools for monitoring multicast topologies have been enunciated [9] but most of these mechanisms are protocol dependent, inflexible and can not easily be generalized to other services or topologies. The mechanisms presented in this paper on the other hand can be used to monitor more general topologies and we give an example of how they have been used to monitor the SLIM network layer multicast protocol.

Our mechanisms inherently implement a control plane that supports many-to-one and one-to-many operations for the purposes of measuring and monitoring general topologies. Essentially our mechanism can be viewed as combining active multicast and active gather-cast [10] to realize the measurement functions.

Some of the ideas behind our work are similar to what is presented in [11]. However, even though all state maintained by our mechanisms is soft and possibly short-lived it is not self destroying in the same sense as defined in [11]. In contrast the state maintained by our mechanisms is explicitly introduced and assumed to exist for a substantial amount of time.

3 Description of the Mechanisms

The mechanisms perform three main functions: 1) local maintenance and information collection from the local service level module, 2) a distribution mechanism to propagate (new query) functions and information from a collection point to the active nodes of the topology, and 3) information gather implemented by a protocol that propagates

information from leaves in the topology towards a designated root applying a (query specific) summation function at each intermediate node. Fig. 1 depicts the gather and distribution mechanisms.

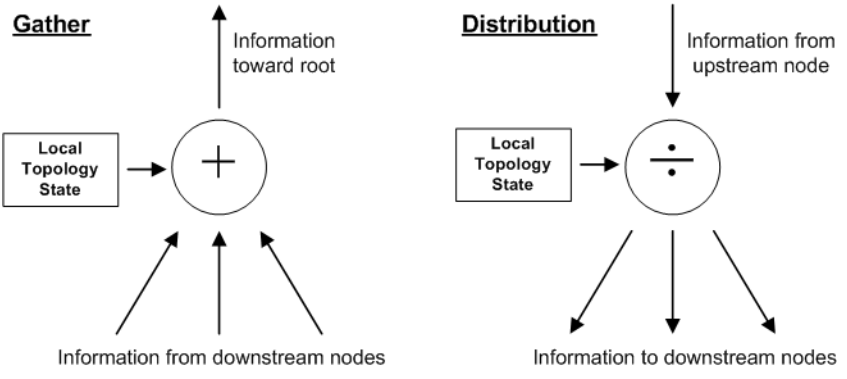

Fig. 1. The gather and distribution mechanisms

We refer to the single source tree rooted at the observer as the collection channel. Each node in the collection channel, apart from the root, has exactly one upstream neighbor and zero or more downstream neighbors. The local state at a measurement node consists of the topology identifier of the channel source, the source specific channel identifier (a single source can have multiple collection channels), a downstream state per each downstream neighbor and the local topology state from the service level module.

The mechanisms implement the collection and computation of three general base attributes that we believe are of sufficient value to most service level topologies, namely the number of leaves of a collection channel, the total number of nodes in a collection channel and the height (or max depth) of the collection channel. The mechanisms define local collection, sum and distribution operations that can be dynamically instantiated on a per service basis or service defined per flow basis. The mechanisms are designed for the control plane and outside of the data forwarding path.

3.1 Local Collection Mechanism

The information collection mechanism is a module that contains the service specific objects for each service level topology the node may be operating. Each object collects the service specific local state from the topology manager running on the node. We define an abstract interface that each service level topology manager implements, enabling it to export its local state to the object performing the local information collection.

```
Interface local_state;
local_state* getLocalState(void);
```

The `local_state` object and the `getLocalState` function must be implemented as part of an adaptation of our mechanisms for a given topology management system. The content of the `local_state` may vary between topology mechanisms, and may for example be different for a network layer multicast topology management, than for an application level overlay.

Our mechanisms support dynamic installation of local information collection modules on runtime, thus adapting and enhancing the local collection abilities of the node. This allows for dynamic installation of objects for new services.

3.2 Distribution Mechanism

The distribution mechanism propagates a query object from the root of the collection channel towards the leaves. The root creates a distribution message and sends a copy on each interface of the virtual topology where the destination address of the message is a collection channel specific identifier *C*. The format of the distribution message, shown in Fig. 2 lower half, consists of some (topology specific) *base* values, a *type identifier*, type specific *data* and the sum and distribution operations for the service type. If a new type identifier is provided the node installs the code for the summation and distribution methods.

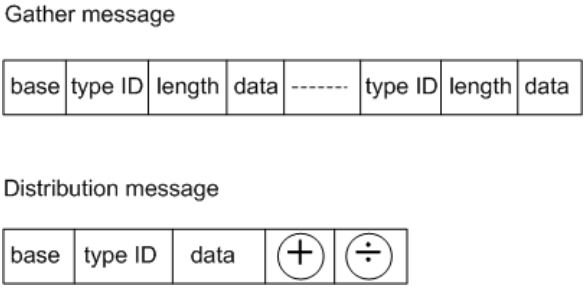

Fig. 2. The gather and distribute messages respectively

A node receiving a query object records the source address of the message as the root of the collection channel and the destination address as the channel identifier. The distribution method specified by the type identifier in the message is then applied to the message. The distribution method may retrieve the *local state* from the topology manager, before producing a new message to be forwarded to each downstream neighbor in the virtual topology. In the simplest case the new message forwarded downstream is the same as the arriving one. In other cases a different message may be forwarded to each neighbor.

From a functional point of view the primary use of the distribution mechanism is to allow the collection point to subscribe to the continually updated computation of results specified by the query object. Since the query object is an arbitrary code (supported by the execution environment of the topology) the range of queries that can be computed is substantial. A query subscription results in the gather process being activated to periodically to compute a distributed time dependent global state

estimation at the collection channel root. A distribution message with a null value for the summation operation will cancel the subscription.

An important use of the distribution mechanism is to distribute information from the root of the collection channel to the nodes of the topology.

3.3 Gather Mechanism

The gather mechanism propagates information from the leaves of the collection channel towards the root by periodically sending updates upstream. At each node the gather involves two functions: a) Receiving and processing gather messages from downstream, and b) preparing the sending a gather result upstream. The two functions are performed asynchronously on each channel.

A gather message received from downstream on a given channel is simply stored as part of the channel gather state, and overrides any previous state on that channel from the same downstream node.

To compute the new gather result, for a given channel, the summation function of each message type associated with a given channel is invoked, each function producing a type specific result, that is appended to the message being prepared. The summation function uses the downstream gather state for each downstream neighbor and the local state from the topology manager.

The format of the gather message is shown in Fig. 2. The message consists of some topology specific set of *base* values (leaves, weight and height) at a node and a segment for each type of collection function associated with the channel (by a previous distribute message). Each service specific segment consists of a service *type ID*, service specific header *length* and service specific *data.*

3.4 Computing the Base Attributes

The general base attributes computed for each collection channel are the number of leaves (l), the total number of nodes (w) and the height (h) of the collection channel.

The number of leaves can be estimated at the root of a collection channel using the gather mechanism. The base field of the gather message contains a leafcount field used by the nodes in the collection channel for this computation. A leaf node sets the value of the leafcount field in the gather message to 1 and sends the gather message towards the source as usual. An upstream node n computes the number of its downstream leaves according as the sum of the leafcounts from all its neighboring downstream nodes. More generally the leafcount at a node n is computed according to

$$l_n = \begin{cases} 1 & if\ n\ is\ a\ leaf \\ \sum_{j \in down(n)} l_j & otherwise \end{cases}$$

where *down(n)* is the set of neighboring downstream nodes of n. This way the leafcount at the root, l_{root}, is simply the current estimate[1] of the number of participants in

[1] Note that due to packet losses, delays in delivery and asynchrony across the topology the result may not be accurate, but is instead a (very good) estimate.

the collection channel. More generally the leafcount at any intermediate node is the number of participants in the subtree rooted at that particular node.

In a similar manner to the estimating of the number of participants the gather mechanism is used to estimate the number of internal nodes in a service level topology. We define the weight of a collection channel as the number of all nodes in the channel, both leaves and internal nodes. The weight at a node n is then given by

$$w_n = \begin{cases} 1 & \text{if } n \text{ is a leaf} \\ 1 + \sum_{j \in down(n)} w_j & \text{otherwise} \end{cases}$$

The height of the collection channel can be computed by finding the subtree with the maximum height and adding one to that value. More generally the height h of any node n in the collection channel can be computed according to

$$h_n = \begin{cases} 0 & \text{if } n \text{ is a leaf} \\ 1 + \max_{j \in down(n)} \{h_j\} & \text{otherwise} \end{cases}$$

To reflect the latest changes in the topology, each node periodically sends a gather message towards the root of the collection channel containing the latest values for l, w and h. The distribution mechanism can be used to propagate the number of participants estimate from the root to all nodes in the collection channel.

3.5 Characteristics and Implementation Assumptions

An important tradeoff in the realization of the gather mechanism is the length of the update period. A short update period gives better state estimates at the increased cost of bandwidth and processing at the nodes. In general the update period should depend on the packet volume of the service being monitored to ensure that the overhead of maintaining the gather state is a some small fraction of the volume of the service flow (say less than 1%) or overall resources. The mechanisms support the dynamic adaptation of the refresh period.

We do not assume reliable delivery of the distribute or the gather messages. Consequently, message loss may affect the effectiveness of the mechanisms. As queries are effectively subscriptions, repeated transmissions of distribution messages can be employed to ensure completeness, potentially resulting in some but inconsequential delay in query distribution.

A more persistent impact is caused by losses of gather messages, resulting in the current estimate at the collection point being a random number continually affected by losses. Our approach described above is designed to minimize the impact of a single packet loss, as the downstream state remains valid (for some significant time) until updated from below. Thus if an update is not received from a particular subtree in round T the update value for that subtree received in round T-1 remains in use.

4 Monitoring and Managing SLIM Multicast

We have used our mechanisms for monitoring and managing SLIM multicast sessions. The local collection mechanism communicates with the topology management module of SLIM. We use the mechanisms to compute estimates of a number of key properties of the multicast topology and monitor changes in these from our service management center. In addition to trigger queries, we utilize the distribution mechanism to trigger the proccess of updating running code in our implementation to enhance and update the monitoring abilities of the active nodes.

4.1 Self Configuring Lightweight Multicast – SLIM

SLIM [12] is a single source multicast paradigm for the Internet that self-configures over the unicast infrastructure by dynamically building network layer IP-in-IP tunnels as required. The SLIM signaling protocol thus constructs and maintains a dynamic service level topology for multicast. A multicast channel in SLIM is identified by the pair $<S,C>$ where S is the channel's source and C is the source specific channel identifier. To create the single source distribution tree a SLIM client sends a JOIN control message towards the source S. SLIM enabled routers intercept the messages and create the appropriate forwarding state in their flow based classifier and construct dynamic tunnels if the JOIN message has been forwarded through routers not supporting the SLIM protocol. When the JOIN message reaches the first router that is already a node in the distribution tree of $<S,C>$ the new branch is added to the distribution tree and the JOIN message suppressed. The only multicast specific functions of SLIM are the control plane topology management, which operates out of data-path and manipulates router classifiers (forwarding table) and tunnel facilities.

4.2 Implementation

We have implemented a monitoring system for SLIM based services using our mechanisms on the Pronto [13] programmable router using packet processors [14]. For the purposes of our implementation the Pronto router provides strong separation between services and protocols implemented in a user space execution environment and the data-path router facilities realized at kernel level. Data-path packet processors furthermore support the composition of paths through the router. In particular, a path can have multiple branches, each branch composed of one or more packet processors. Thus branches may differ in functionality. For our mechanisms this allows us to monitor the information volume sent on individual multicast branches by creating packet/byte count packet processors for each branch.

The monitoring mechanisms are implemented as a user-space daemon. The local collection object communicates with the Topology Management (TMP) daemon of SLIM to collect the local state information. We have implemented this through the use of shared memory. The TMP daemon maintains a table in shared memory that contains the local state for each active multicast flow. The local collection object creates a read-only instance of the class interfacing the shared memory upon which it can invoke a `getLocalState` method as described in Section 3.1.

Each SLIM channel corresponds to a collection channel where the root of the collection channel is the SLIM source, *S*. For each collection channel the local topology state consists of the number of active downstream branches and the system unique identifier of each branch. The local collection state and the state from each downstream node is used to compute the number of leaves, weight and height of each channel using the gather mechanism. Each gather message consists of the base values for *l, w* and *h* and in addition the number of packets and bytes received at the node for the flow identified by <*S,C*>. The base values at the root of the collection channel can be used to estimate the number of multicast receivers (leaves), longest path to a receiver (height) and the number of internal nodes in the multicast channel as $w - l$. Using the byte/packet values in a received gather message a node can estimate the link lossrate of each downstream link by comparing the downstream value with the number of bytes/packets received.

An initial distribution message is simply sent on the multicast channel being monitored. The distribute and gather messages are sent with the router alert IP-option and a special protocol ID which results in the active nodes intercepting the messages and dispatching them to the module implementing the mechanisms. The distribution mechanism can be used to distribute and update running code in the active nodes. Our implementation is in C++. The C++ code for the local collection, gather and distribute objects of the monitoring daemon can be introduced and updated through the use of dynamic C++ classes [15].

4.3 Bottleneck Discovery – Placement of Active Retransmission

Using our mechanisms discovering bottleneck links and deploying active retransmission is relatively easy. Each node transmits upstream the number of packets it receives on a given channel. By comparing this value to the local observation of packets received, the gather computation reveals if excessive losses are occurring on any of the downstream links. If so it deploys active retransmission on that particular link (using the Pronto packet processors this is very easy to do on a per branch basis).

5 Monitoring More General Virtual Topologies

In this section we give examples of how the mechanisms can be used for general service level topologies. A meshed topology structure does not have a distinguished root and has multiple paths between nodes in the topology. However, from any node a virtual topology a well defined (minimum cost) spanning tree will typically exist. A collection point initiates query processing by sending a distribute message over such a spanning tree.

Although a wide range of functions can be computed over general topologies using the mechanisms, computing link attributes is more difficult than computing node properties, as a spanning tree will visit all nodes but will not traverse all links. Of course this can be overcome, but the mechanisms do not provide explicit support to address this issue.

5.1 Monitoring SLIM Router Deployment Using a Control Plane Overlay

As part of our research on multicast we have been offering televisions distribution services and teleconferencing experimentation over SLIM multicast for over a year now. Although the number of SLIM-enabled routers is still small their number is growing. To better exploit available SLIM routers, to keep track of their distribution, to update the SLIM code, and as part of our ongoing research on advanced group management the SLIM protocol now supports a control plane overlay.

By building a spanning tree from our local SLIM router, we can use our mechanisms over this overlay, to keep track of the number of routers, the diameter and density of the deployment, as well as to facilitate distribution of code updates. This is in addition and separate from the flow level monitoring described in Section 4.

5.2 Applications to Service Level Routing

In [2] Anerousis et al employ a virtual topology of dynamically constructed tunnels to route requests to a named service realized by a virtual host that, in theory, provides the service. A virtual host has an IP address and appears to the rest of the Internet as a regular host. A request from a client is routed to a particular (physical) server by a set of service level (application level) nodes. The routing is determined in real time through the service level routing map and may take into consideration user attributes such as originating address, and network and server attributes such as load. Rather than relying on modified DNS based redirection schemes at the edges of the network, in the service level routing of [2] the service level nodes use service semantics, and load and availability attributes to transparently routes service requests to the appropriate servers based on a variety of criteria.

Requests are routed over a layered virtual topology. Client requests are routed by the IP infrastructure to the service level router (SLR) closest to the client (using standard destination based routing). The packets are then directed to an SLR one layer up using IP-in-IP tunnels constructed dynamically if needed. This continues until the SLR of a particular hosting site is reached. The SLR at the hosting site further tunnels the packets to the host that is best suited for serving the request. Each server host terminates the tunnel and recovers the original datagram exactly as it was sent from the client. From the addresses in the original datagram the receiving server process learns the client address as well as a the virtual host address. Acting as the virtual host, it transmits its replies directly to the requesting user client, using the address of the virtual host as its source address, and avoids the service level virtual structure. The multiple levels improve scalability and load balancing effectiveness.

In the service level routing topology the availability and load of the service level routers, and the server hosts play a dominant role in providing consistent dependable service quality. Given appropriate policies to determine the local load metrics at the SLR's or the servers, we employ our active mechanisms to propagate and update the availability and load information in the SLR topology as follows. Each lowest layer SLR is a collector that initiates a collection channel. The SLR at each hosting site joins the collection channel of each leaf. At each hosting site the SLR computes a

load metric for each virtual host hosted at the site, and propagates using the gather mechanism. Intermediate SLR's collate the load metrics from below, combine them with a network load metric, and compute a load metric for the downstream tree that are propagated upstream. Each lowest layer SLR thereby receives a metric of load from each branch that it uses to perform load aware route selection (combined with other criteria).

6 Discussion

Insensitivity to non-cooperating nodes. In the heterogeneous Internet assuming uniform deployment is unrealistic. Even under active networking assumptions, homogeneity cannot be assumed, as nodes may vary in their capabilities, authentication policies, and access given to installed services. While the correct operation of our mechanisms does not require uniform cooperation across the virtual topology, the effectiveness of the mechanisms is reduced. As the ratio of non-cooperating nodes in the topology increases, it may become attractive to employ some of the techniques of [3,4] to infer the properties of the non-participating segments of the virtual topology. The same applies if physical topology attributes are of interest as metrics for a virtual topology that still consists only of relatively few nodes of the physical topology.

It is relatively easy to determine the density of the virtual topology over the physical one, by tracking the TTL count between virtual hops, and summing up all physical hops over a given channel. At a given collector, the density can then be defined as the physical hop count over the weight of the tree.

IP as a service level topology over the transport network. In traditional network operation models the physical network (e.g. the optical transport network) is viewed as providing physical transport to a number of service networks running over virtual topologies on top. In this model, IP is just another service constructing a service specific virtual topology. Alternate models assume that the routers manage the underlying physical resources [16]. Our mechanisms are agnostic to this and are suitable for such an environment by deploying our mechanism in IP routers, and could then provide the collection mechanisms described above for the IP network.

7 Conclusion

In this paper we have described extensible general purpose mechanisms to monitor and measure characteristics of a service level topology at the nodes of the topology. The mechanisms provide means to dynamically deploy a distributed observation function to the nodes of the topology and to collate the observations into a result given to the requestor on a subscription channel. The value of the mechanisms is validated by their extensive use in our experimentation with multicast. In addition we have given examples from our experimentation with the same mechanisms over general service topologies, including overlays for router discovery, and service level routing.

References

[1] N.G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan, and J. E. Van der Merwe, "A Flexible Model for Resource Management in Virtual Private Networks", IEEE/ACM Transactions on Networking, No.5, Oct. 2002.

[2] N. Anerousis and G. Hjálmtýsson, "Service Level Routing on the Internet," in proceedings of Globecom'99, Rio de Janeiro, Brazil, December 1999.

[3] A. Adams, T. Bu, R. Caceres, N.G. Duffield, T. Friedman, J. Horowitz, F. Lo Presti, S.B. Moon, V. Paxson, and D. Towsley. "The Use of End-to-End Multicast Measurements for Characterizing Internal Network Behavior", IEEE Communications Magazine, May 2000.

[4] T. Bu, N. Duffield, F. Presti, and D. Towsley, "Network tomography on general topologies," in proceedings of ACM SIGMETRICS, 2002

[5] Y. Chen, D. Bindel and R. H. Katz, "Tomography-based Overlay Network Monitoring", in proceedings of the 2003 ACM SIGCOMM conference on Internet measurement, Miami Beach, USA, 2003

[6] S. Ratnasamy and S. McCanne, "Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-end Measurements," in proceedings of IEEE Infocom'99, Mar. 1999.

[7] W. Cui, I. Stoica, and R. H. Katz., "Backup path allocation based on a correlated link failure probability model in overlay networks". In the proceedings of ICNP 2002, November 2002.

[8] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, Robert Morris, "Resilient Overlay Networks", in Proc. 18th ACM SOSP, Banff, Canada, October 2001.

[9] K. Sarac and K. Almeroth, "Supporting Multicast Deployment Efforts: A Survey of Tools for Multicast Monitoring", Journal of High Speed Networking - Special Issue on Management of Multimedia Networking, vol. 9, num. 3/4, pp. 191-211, March 2001.

[10] K. L. Calvert, J. Griffioen, B. Mullins, A. Sehgal and S. Wen, "Concast: Design and Implementation of an Active Network Service", IEEE Journal on Selected Area in Communications (JSAC). Volume 19, No. 3. March, 2001.

[11] S. Wen, J. Griffioen and K. L. Calvert, "Building Multicast Services from Unicast Forwarding and Ephemeral State", Computer Networks: the International Journal of Computer and Telecommunications Networking. Elsevier Science. Vol.38, Issue 3. February, 2002. pp.327-45.

[12] G. Hjálmtýsson, B. Brynjúlfsson and Ó. R. Helgason, "Self-configuring Lightweight Internet Multicast", accepted for publication at IEEE SMC 2004, Netherlands, october 2004.

[13] G. Hjálmtýsson, "The Pronto Platform - A Flexible Toolkit for Programming Networks using a Commodity Operating System," in the proceedings of OpenArch 2000, Tel Aviv, Israel, March 2000.

[14] G. Hjálmtýsson, H. Sverrisson, B. Brynjúlfsson and Ó. R. Helgason, "Dynamic packet processors - A new abstraction for router extensibility", in proceedings of OPENARCH-2003, San Francisco, April 2003.

[15] R. Gray and G. Hjálmtýsson, "Dynamic C++ classes - A Lightweight mechanism to update code in a running program," in proceedings of the USENIX Annual Technical Conference, pp. 65-76, June, 1998

[16] A. Greenberg, G. Hjálmtýsson and J. Yates, "Smart Routers - Simple Optics. A Network Architecture for IP over WDM," in the proceedings of the OFC 2000, Baltimore, March 2000.