VMScope – A Virtual Multicast VPN Performance Monitor

Lee Breslau[*], Chris Chase[†], Nick Duffield[*],
Bill Fenner[*],Yanhua Mao[‡], Subhabrata Sen[*]
[*]AT&T Research, [†]SBC Labs, [‡]University of California San Diego
{breslau,duffield,fenner,sen}@research.att.com, chase@labs.sbc.com, maoyanhua@cs.ucsd.edu

Keywords

Virtual Private Networks (VPN), Multicast, Multicast VPN, Performance Monitoring

ABSTRACT

The growth of one-to-many applications in enterprise networks is fueling the demand for VPNs to support multicast applications. The deployment of such a Multicast VPN service creates the need for appropriate management tools and techniques including performance monitoring, problem isolation and troubleshooting. In this paper we present the MVPN monitoring problem, identify requirements for solutions to the problem, and describe VMScope, a virtual MVPN performance monitor designed to facilitate provider monitoring and debugging of the MVPN service.

1. INTRODUCTION

Enterprise networks are increasingly turning to Virtual Private Networks (VPNs) to connect geographically disparate locations. In a provider-based VPN, rather than connecting customer sites together using dedicated private lines, each customer site connects to one or more edge routers in the provider network. Customer traffic is encapsulated and carried across the provider network, decapsulated by a remote provider edge router and handed off to the customer router. Traffic belonging to different customers is segregated in the provider backbone, and the provider network is opaque to the customer. Such provider-based VPNs provide a scalable and secure way for a service provider to support many customers across its backbone. Large VPNs can connect hundreds of customer locations.

In many respects, the applications supported in large enterprise networks mirror those seen in the wider Internet. These include the web, email and instant messaging, among others. In addition to these, there is a growing need within enterprise networks to support a class of applications that benefit from one-to-many network transport, such as all-employee broadcasts and software distribution. Such applications are driving the deployment of IP multicast within enterprise networks. Multicast was first deployed experimentally in the Internet more than 15 years ago [2]. However, due to scalability, security and business issues, multicast has yet to be widely deployed and used in the Internet [3]. These issues are much more easily addressed in enterprise networks.

The growth of one-to-many applications in enterprise networks coupled with the reduced barriers to deployment in this environment are driving the deployment of multicast within private networks. Since these private networks are increasingly being supported with provider-based VPNs, there is thus growing demand for VPNs to support multicast applications. The deployment of such a service brings with it the need for appropriate management tools and techniques including performance monitoring, problem isolation and troubleshooting. This paper presents a first effort at developing such a tool.

The predominant method for supporting provider-based VPNs uses MPLS as the encapsulating technology across the provider backbone [8]. These VPNs provide only unicast transport across the backbone and as originally deployed were not amenable to supporting customer multicast traffic. Extensions to these MPLS-based VPNs have been defined which enable the deployment of Multicast VPN (MVPN) service [7]. In an MVPN, customer multicast traffic is encapsulated at an ingress provider edge router and delivered to one or more egress provider edge routers.

Managing generic IP multicast has traditionally been difficult [3, 10]. Managing MVPN service presents several additional challenges for a service provider. In an MVPN, multicast exists in two domains – the customer and the provider – and an encapsulation mechanism binds the two together. The set of multicast groups in both domains is dynamic, as is the mapping between them. Further, the set of group members (again in both domains) can change, implying that the distribution trees for each group may also change. Within the provider backbone, the multicast trees can be quite large, including hundreds of routers. Performance monitoring, problem isolation and troubleshooting are challenging in such an environment.

In this paper we present the MVPN monitoring problem, identify requirements for solutions to the problem, and describe VMScope, an MVPN performance monitor designed to facilitate provider monitoring and debugging of MVPN service. VMScope uses tunneling technology to establish virtual connections to routers in the provider network and uses these tunnels to enable remote monitoring of customer

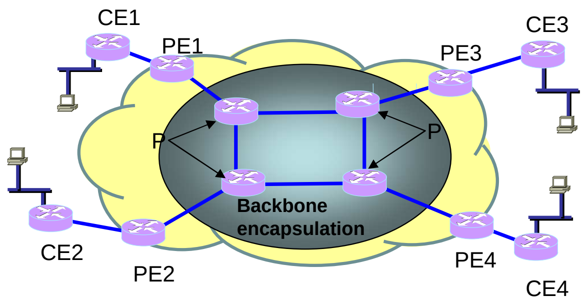

Figure 1: Multicast VPN

multicast performance. In particular, from a single monitoring host, VMScope can send and receive multicast traffic across the provider backbone. Within the provider network, this traffic appears like, and hence is treated the same as, multicast traffic sent by the customer into the MVPN. Thus, VMScope enables the provider to measure customer performance across the MVPN and provides a critical performance monitoring and troubleshooting tool. While there exist techniques for monitoring generic IP multicast (see [9] and references cited therein), we are not aware of any existing tools for MVPNs. Although VMScope represents an important contribution, refinement and evaluation of it are ongoing. There are many interesting open research problems in this space, presenting an opportunity for future contributions by the community.

The rest of this paper is organized as follows. In Section 2 we present a brief overview of the technology used to support MVPNs. We then describe VMScope in Section 3 and report on our evaluation of, and experience with, it to date in Section 4. We conclude with a discussion of extensions, open issues and ongoing work in Section 5.

2. MVPN OVERVIEW AND MONITORING REQUIREMENTS

In this section we present a brief overview of MVPN. In the interest of space, we refer readers to [7] for the complete technical details of the service, and confine ourselves here to those aspects needed to motivate the need for, and understand the operation of, VMScope. We then identify the challenges in and requirements for designing a system to monitor MVPN performance.

2.1 Multicast VPNs

An MVPN (Figure 1) includes the same functional components as a unicast VPN. At each customer location, a customer edge (CE) router connects to a provider edge (PE) router. The PE routers are responsible for encapsulating and decapsulating packets coming from and going to the CE routers. Provider (P) routers forward packets within the provider network and are not involved in VPN-specific operations.

Unicast VPNs use MPLS as the encapsulating protocol. The ingress PE router attaches an MPLS label to a packet it receives from an attached CE. This label is used to forward the packet to the egress PE, where it is decapsulated and passed to the attached CE. An MVPN transports a customer multicast packet across the provider network to relevant destination(s). The customer multicast packet may be destined for multicast group members at more than one customer location. Hence, a point-to-point delivery mechanism across the provider network that delivers the customer packet to only a single egress PE is insufficient. Rather, a point-to-multipoint transport mechanism is needed.

IP multicast, which provides such a transport mechanism, is (not surprisingly) used to support MVPN service. When a PE receives a multicast packet from a CE for delivery across the MVPN, the PE uses Generic Routing Encapsulation (GRE) [4] to encapsulate the customer packet in another multicast packet. This new multicast packet is sent across the provider backbone to one or more PE routers. Transmission across the backbone is entirely transparent to the customer; the customer merely sees a logical connection across the provider cloud. At the receiving PE routers, the outer header is removed, and the original customer multicast packet is transmitted to the attached CEs. At this point, the customer domain multicast resumes. That is, the original multicast packet is delivered to group members in the customer domain downstream from the CE.

This encapsulation and distribution is abstractly quite simple. However, several design issues must be addressed to achieve the desired functionality. Relevant questions include: 1) How does the ingress PE select the multicast group used in the provider domain for encapsulation? 2) How does the encapsulated packet reach the right set of egress PEs?

Within the provider backbone, unique multicast addresses are used to encapsulate traffic for each MVPN. This assures that traffic from different customers is segregated in the backbone. In particular, each customer is assigned a default multicast group, referred to as a Default MDT (for Multicast Distribution Tree). All PE routers that participate in a given MVPN know this group address (e.g., by configuration.) All of the PEs join the multicast group, and consequently receive all packets sent to the multicast group.

The Default MDT provides a mechanism for a packet sent by any PE in an MVPN to reach all other PEs in the MVPN. A customer multicast packet is encapsulated in a provider multicast packet whose destination is the Default MDT address. Generic multicasting routing (such as provided by PIM or other multicast routing protocols) delivers the packet across the backbone to all other PE routers, by virtue of their having joined the Default MDT group. The receiving PE routers decapsulate the packet and can deliver it to their attached CE routers.

In those cases when the customer multicast group does not have group members at every customer location, the Default MDT delivers the packet to more PE routers than actually need it. Those PE routers that are attached to CE routers which have no downstream group members, drop the packet instead of forwarding it to the CE.[1]

2.2 Monitoring Goals/Requirements

While the description above is necessarily brief, the resulting MVPN system is fairly complex. A service provider may support hundreds or eventually thousands of MVPN

[1]While data is only delivered to customer routers with downstream group members, this still wastes bandwidth in the provider backbone. The MVPN specification allows for special *Data MDTs* to encapsulate high bandwidth groups. PEs only join a Data MDT if there are group members downstream from its attached CE, thereby eliminating the wasted bandwidth.

customers, each spanning up to hundreds of sites. It will be incumbent on the provider to be able to monitor the performance of these MVPNs, and when performance falters, to be able to identify the source of problems. The mechanisms and protocols used in the provider backbone to implement unicast VPNs and MVPNs are quite different. A problem that affects one domain (e.g., unicast) may not impact the other (e.g., multicast), and vice versa. Hence, traditional tools for monitoring unicast VPNs are inadequate; new kinds of tools tailored to identify multicast-specific issues are needed to support MVPN service.

As this is a new networking technology, there are undoubtedly monitoring requirements (and their eventual solutions) of which we are as yet unaware. However, as an initial attempt to address this problem, monitoring technology that allows the provider to assess the performance that a customer would receive across the MVPN will certainly be required. In particular, we focus on end-to-end loss rate and delay as metrics relevant to the customer (and are likely to be embodied in service level agreements.)

We designed and built our MVPN performance monitor with the following requirements in mind. First, the monitor must be implemented entirely in the provider network and must be transparent to the customer. That is, the customer should not be directly aware of or impacted by the monitor. Second, the monitor must be easy to deploy. That is, it should not require special purpose hardware, and the cost to deploy it should be minimized. A third (and related) requirement is that it should be easy to manage. Management costs can often be substantial and may in fact outweigh deployment costs. Fourth, the monitor must be compatible with currently deployed routers. That is, it must exploit existing features and protocols of today's routers and must not depend on any experimental or as yet unimplemented router features. Fifth, the monitor must be flexible in its ability to start and stop monitoring on a per-MVPN, and within an MVPN on a per-PE, basis. That is, monitoring need not be continuous, but may in fact be an activity engaged in an on-demand fashion. Finally, the monitor must be scalable in the sense it can monitor many and large MVPNs.

3. VMSCOPE - VIRTUAL MULTICAST MONITORING TOOL

VMScope is a virtualized performance monitor that enables remote monitoring of MVPNs. It is able to measure packet loss rates and delays seen by customer traffic as it traverses the MVPN. It uses standard features found on routers to send and receive probes that follow the same path across the backbone as customer MVPN traffic. Further, it is deployed on a single monitoring station. As such, it provides a low cost and easily deployable way to provide monitoring not otherwise available in this challenging new environment. In this section we describe the VMScope architecture and our implementation of it.

3.1 Architecture

VMScope consists of software running on a single monitoring host that communicates with routers in the network. In the current discussion, we assume the routers are PEs, but there is nothing the precludes VMScope more generally from being used with P routers as well. VMScope maintains a virtual tunnel (via GRE encapsulation) with each PE router that it will monitor. A GRE tunnel enables either end to encapsulate an arbitrary packet (referred to as the *inner* packet) in another packet (referred to as the *outer* packet) and transmit it to the other tunnel endpoint where the encapsulation is stripped and information in the inner packet is used for further processing/forwarding.

The key features of GRE tunneling that we exploit are 1) the inner packet is not processed by routers between the two tunnel endpoints, and 2) the tunnel implements a virtual IP link allowing the two tunnel endpoints to appear directly connected in the IP topology. Other tunneling technologies that meet these criteria would have also sufficed, and we note that ATM virtual circuits were employed in a previous system that used a single monitoring host to measure delays through a unicast network [1].

Given a tunnel between a monitoring host, M, and a remote router, $R1$, we can employ the GRE features as follows. First, M can send IGMP [5] join messages to $R1$ across the tunnel for an arbitrary multicast group, G. $R1$ will join the multicast tree for G (using whatever multicast routing protocol is in use.) When $R1$ subsequently receives a multicast packet addressed to G it will forward that packet across the tunnel to M. The GRE-encapsulated packet is forwarded as a normal unicast packet, and it does not require routers on the path between M and $R1$ to support multicast.

In our second use of the GRE tunnels, M can send multicast packets addressed to an arbitrary multicast group, G, across the tunnel to $R1$. The tunneled packets are forwarded across the tunnel as unicast packets, and do not require the routers on the path between M and $R1$ to support multicast. When $R1$ receives such a packet, it removes the encapsulating header and forwards the original multicast packet as dictated by its multicast forwarding table.

VMScope uses these building blocks as follows. Given a monitor, M, routers $R_1 \ldots R_n$, tunnels $T_1 \ldots T_n$ to each R_i, and a multicast group, G, M transmits IGMP join messages for G across T_i, for $2 < i \leq n$, and transmits multicast packets addressed to G across T_1. R_1 decapsulates the packets it receives from M and transmits them as normal multicast packets. Given the IGMP join messages received by each R_i, $2 < i \leq n$, the multicast routing protocol will construct a distribution tree for G such that packets addressed to G and forwarded by R_1 will reach the other R_i. When a router, R_i, receives a multicast packet addressed to G, it encapsulates the packet using GRE and forwards it across the tunnel T_i to M. Thus, M injects multicast packets into the network and receives copies of these packets back. Specifically, for each packet transmitted to R_1, M receives $n-1$ copies back (modulo packet loss.)

The packets injected by M include an application layer header which contains a sequence number. In addition, M stores the sending time of each packet. For each packet that M receives, the router R_i which forwarded it to M is also known (by virtue of the GRE tunnel over which it arrived.) Hence, M can compute both the loss rate and end-to-end delay on the path through the network via each R_i.

The system described above enables a monitor to inject multicast packets into a network and receive one or more copies of those packets back from the network. This is well-suited to monitoring MVPNs. Recall from Section 2 that there is a multicast group, referred to as the Default MDT, associated with each MVPN. Every PE that participates in the MVPN joins and maintains continuous membership in the Default MDT.

Figure 2 depicts a typical VMscope deployment scenario. In order to monitor the Default MDT, the monitor M establishes a GRE tunnel with each PE participating in the MVPN. M joins the Default MDT over each of these tunnels, and also sends periodic probe packets across each tunnel.[2] Thus, the monitor can compute the delay and loss rates between every pair of PEs in the MVPN. Since the probe packets travel on the *same* multicast distribution tree as the MVPN packets sent by the customer (e.g., CE-A in the figure) on the Default MDT, the monitor is able to measure performance experienced by the customer and can easily identify and locate performance problems.

VMScope monitors performance across the Default MDT by injecting probe packets addressed to the Default MDT group. While the probe packets follows the same path as customer traffic through the provider backbone, it is critical that they be segregated from the actual customer traffic at the edge of the network. This translates into four requirements: 1. Customer packets received by a PE must be forwarded to attached CE(s) according the MVPN specification. 2. VMScope probe packets received by a PE must *not* be forwarded to attached CE(s). 3. VMScope probe packets received by a PE must be forwarded back to the monitor. 4. VMScope must not process customer packets.

The first and third requirements are trivially met. Customer packets received by a PE are handled according to the MVPN specification and are thus forwarded to attached CE(s) according to the MVPN forwarding rules. With regard to forwarding probe packets back to the monitor, the IGMP join messages issued by the monitor across the tunnel will cause all multicast packets addressed to the Default MDT to be forwarded across the tunnel. The fourth requirement is also met quite easily. All multicast packets destined for the Default MDT (VMScope probes as well as customer packets) will be forwarded across the tunnel back to the monitor. However, the two kinds of packets can be distinguished quite easily (see below), and the monitor can drop customer packets without processing them.[3]

The second requirement, perhaps the most critical, also follows from the MVPN specification. Recall from Section 2 that multicast packets received from a CE by an attached PE to be forwarded across the Default MDT are encapsulated using GRE.[4] According to the MVPN specification, only the GRE encapsulated customer packets are forwarded to the attached CE routers (after their outer headers are removed.) The VMScope probes, which once inside the network are no longer GRE encapsulated, are not forwarded on the PE-CE link.

Before describing our implementation of VMScope we address one final design issue. We stated earlier that VMScope computes loss and delay measures through the network. However, the path measured by VMScope, $M \rightarrow PE_i \rightarrow PE_j \rightarrow M$, where PE_i and PE_j are the ingress and egress PEs, respectively, includes the two tunnels between the monitor and the PEs. We want to exclude these components from our measurements as the focus of our monitoring is on the path through the provider network taken by customer packets (i.e., $PE_i \rightarrow PE_j$.)

To account for the tunnel components of the end-to-end path, VMScope measures round trip delay and loss on between the monitor and each tunnel endpoint (using *ping*.) As the VMScope probe packets only traverse a tunnel in one direction, we make the simplifying assumption that loss and delay are symmetric on the two paths and then compute one-way loss and delay based on the ping measurements.[5]

3.2 Implementation

VMScope has been implemented as a user level process using raw sockets and *libpcap*. This required that modules supported by many operating systems be implemented in the application. However, eliminating issues related to OS-specific implementation of the multicast API and GRE tunnels resulted in more portable code with increased flexibility. While we did need to address cross-platform issues concerning raw sockets, confining all the portability issues to one module was easier than dealing with them more generally. The system architecture of VMScope is shown in Figure 3.

The raw socket interface is used to send packets, and provides the flexibility to create all the necessary packets and headers (IGMP, GRE, etc.) We recorded the sending time of each packet in user space. While the resulting end-to-end delay measurements include a component of host operating system delay, this delay (on a fast machine that is not overloaded) is likely to be very small relative to the network propagation and queuing delays that we are primarily

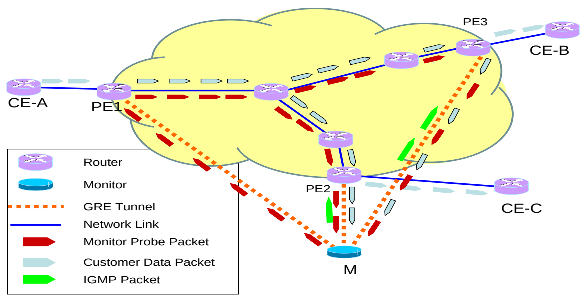

Figure 2: VMScope Operation

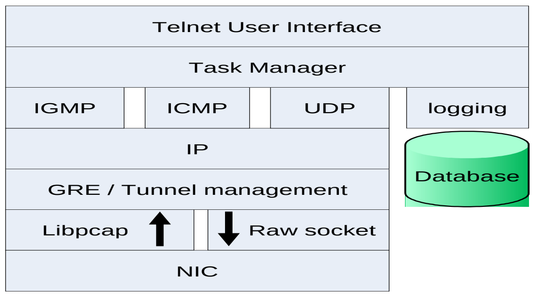

Figure 3: VMScope Software Architecture

[2]The description above assumed a single sender and $N-1$ receivers, but the scheme is fully generalizable to N senders and receivers.

[3]The overhead of transmitting these customer packets to the monitor may be a concern. We address this issue in Section 5.

[4]This instance of GRE encapsulation should not be confused with the GRE encapsulation VMScope uses to transmit packets between the monitor and the PEs.

[5]Obviously, loss and delay on the tunnels may not be symmetric. We return to this question in Section 5.

interesting in measuring. Further, the tunnel specific measurements that are used to remove the components of end-to-end delay contributed by the tunnels also include this host component. Hence, the contribution is removed from the primary measurements with which we are concerned.

While we could have used raw sockets to receive packets as well, we opted instead to use the packet capture library, *libpcap* primarily for two reasons. First, use of libpcap allows us to timestamp received packets as early as possible. Whereas we expect operating system delays on the sending side to be minimal, we expect higher and more variable operating system delays with input processing, as packets may wait on input queues to be delivered to higher layer protocols. Second, libpcap provides an efficient filter mechanism to drop unwanted packets early. Recall that the monitor receives customer packets across the tunnel that it does not process; the sooner that traffic is dropped, the better.

GRE tunnels are implemented above the raw socket and libpcap interfaces. VMScope maintains internal state for each tunnel (inner and outer IP addresses to use), and sets header fields appropriately on outgoing packets. In addition, it can easily identify which tunnel an incoming packet was received over. Because we implemented GRE in the VMScope application (rather than using an operating system implementation), we must also implement the higher layer protocols in the application. These include the IP layer (corresponding to the inner packets), and three higher layer protocols: IGMP, ICMP and UDP. IGMP is used by VMScope to transmit multicast join messages to the routers it is monitoring. ICMP is needed to support the round trip measurements between the monitor and each of the routers. UDP is needed to support the active probe packets transmitted and received by the monitor. For each protocol we need only a subset of functionality and implement those portions. For example, the duplicate membership report suppression mechanisms needed to make IGMP scalable on a multi-access network are unnecessary on virtual point-to-point links (i.e., GRE tunnels) connecting a single router and host.

VMScope runs as a privileged daemon. The initial implementation is controlled by a telnet interface that supports commands to add and drop tunnels, join specified groups on a per tunnel basis, and send packets to a group over a specified tunnel (at a given sending rate). A Task Manager presents a higher layer abstraction to facilitate expected monitoring activity. It enables a group of tunnels to be controlled together so that probing from N senders to M receivers is treated as a single action. Other commands allow the user to retrieve statistics related to the operation of a tunnel or the performance metrics collected on a per group basis. In addition, metrics for each received packet are logged to a database to support additional analysis.

4. VMSCOPE EVALUATION

In this section we evaluate VMScope relative to our initial requirements. Our first requirement was that VMScope be implemented entirely in the provider network and be transparent to the customer. As described in Section 3, we have achieved this. Probe traffic injected by the monitor does not leak into the customer domain. The one way that VMScope might impact the customer is by the additional load it places on the network. The rate of VMScope probes is tunable, and we envision that the monitor traffic will be only a small fraction of the traffic sent by the customer.

The second and third requirements were that the monitor be easy to deploy and manage. The monitor runs on a single general purpose workstation and is hence not expensive for a service provider to deploy. Further, a single centralized monitoring station reduces the management cost relative to a distributed system requiring deployment at multiple network locations.[6]

The fourth requirement was that the monitor be compatible with current router technology. The monitor transmits IGMP, UDP and ICMP packets over GRE tunnels to the routers. These protocols are all standard features of today's commercial routers. Establishing the GRE tunnels requires a few lines of router configuration. Once the tunnels are established at the routers, the monitor can dynamically join and leave groups over the tunnels, and can start and stop sending probe packets. This satisfies the fifth requirement that the monitor be flexible and allow dynamic testing.

Assessment of the final requirement, that the monitor be scalable, and of its general correctness necessitate experimentation. Specifically, we set out to answer the following questions. First, does the monitor perform correctly, in the sense that it is able to accurately measure end-to-end delay and loss through the network? Second, is it scalable, in the sense that it can monitor many large MVPNs? We deployed VMScope in a small laboratory setting in order to address these questions. We describe initial experiments below.

Our testbed consists of three CISCO 7200 routers connected by 100Mb Ethernet and a single 3.2GHz PC running FreeBSD 5.4. Two routers, configured as PEs, were connected by the third, which was configured as a P router. This limited environment is by itself insufficient to evaluate the ability of VMScope to monitor MVPNs with many PEs. To address this shortcoming we configured multiple parallel tunnels between the PC and each of the two PEs. From the standpoint of the monitor, two parallel tunnels to a single router are indistinguishable from tunnels to two distinct routers, so this allows us to evaluate the ability of VMScope to monitor many PEs in an MVPN.

The first set of experiments was intended to validate the correctness of the VMScope implementation. We configured 5 parallel tunnels between the PC and each of the edge routers. Probe packets were sent on each of the tunnels to the first PE and received over the tunnels from the other PE. We used Dummynet [6] on the PC to induce controlled loss and delay for received packets on a per tunnel basis. Table 1 shows the induced and measured loss rates and delays for each of the 5 tunnels. As can be seen, the results are consistent with the expected operation of VMScope. Measured delay is slightly higher than the induced delay, due to actual propagation and transmission delay through the testbed. Measured loss rates vary slightly from the induced loss rates, which can be attributed to the statistical nature of the Dummynet filter.

In an actual deployment, VMScope might be called upon to maintain tunnels to many routers simultaneously, and though the probing rate for any given tunnel and MVPN need not be large, in the aggregate the monitor may need to process many packets (from a large number of tunnels

[6]Deploying a monitor within every network PoP might be feasible. However, this would not be sufficient to identify all network problems (e.g., those between a backbone and edge router.)

Tunnel number	Target loss rate	Measured loss rate	Induced delay	Measured delay
1	2%	2.1%	10ms	10.5ms
2	0%	0%	20ms	20.5ms
3	5%	4.9%	40ms	40.5ms
4	0%	0%	80ms	80.5ms
5	10%	10.1%	100ms	100.4ms

Table 1: Target vs. Measured Delay and Loss from VMScope Experiments

and MVPNs.) Thus, we performed a second set of experiments to assess the scalability of the monitor. For these experiments we removed the Dummynet filters and varied the number of tunnels to each router, the rate that probe packets were sent on each tunnel, and the size of the probe packets. In the interest of space, we omit a full description of the experiments and their results and merely highlight a few key points. We carried out experiments with up to 100 tunnels between the monitor and routers (emulating a scenario in which VMScope would monitor an MVPN with 100 PEs.) We observed that VMScope was able to process 25K packets per second, and achieved an aggregate throughput of 74 Mbps. In these experiments, the switch and routers became the bottleneck before we saturated the monitor running on the PC, so these numbers only provide a lower bound on VMScope performance on this hardware.

While the initial results from testing VMScope are promising, additional evaluations are ongoing. In addition to continued testing of VMScope under a wider range of laboratory conditions, we have also deployed it (again with only two routers) in a production network.

5. CONCLUSIONS

In this paper we described VMScope, a monitor designed to assess the performance of Multicast VPNs. VMScope is a novel tool using virtual tunneling to enable a service provider to directly assess the experience of customer MVPN traffic across its backbone. We have implemented VMScope and our initial experiments indicate that we have achieved a scalable design. While additional work is required before using this tool in a production setting, experience to-date is very promising. Before outlining additional issues that need to be addressed, we first make some additional observations about the use of the tool.

First, we described VMScope as an active monitoring tool that measures its own probe traffic transmitted through the network. We observe that VMScope can also be used in *passive* mode. In passive mode, VMScope receives and measures customer traffic, rather than injecting its own probe packets into the network. Second, while VMScope was designed to monitor MVPNs, it can also be adapted to monitor vanilla (i.e., non-VPN) multicast. Some modifications are required to use VMScope in either of these ways (passive mode, native IP multicast), but we believe these modifications are relatively minor. We omit a full description of these changes in the interest of space.

While our initial results are promising, several issues, which are the subject of ongoing study, need to be addressed before deploying VMScope operationally. The first concerns the bandwidth overhead it imposes on the network. By joining the Default MDT group over tunnels to multiple PEs, the monitor will receive multiple copies of all its probe packets, as well as multiple copies of all customer packets transmitted on the MVPN. For a large MVPN this overhead could be substantial. However, the MVPN specification provides a mechanism (see Footnote 1) to allow high bandwidth customer groups to be encapsulated in special Data MDTs in the provider backbone. This feature provides an engineering mechanism to limit the bandwidth on the Default MDT. The amount of traffic carried on Default MDT groups is in part subject to provider control (i.e., the threshold for characterizing a group as high bandwidth is tunable) and can be studied as MVPN service is deployed further. Additionally, it may be possible to prevent the customer traffic from being transmitted back to the monitor by applying firewall rules at the PE end of the tunnel to keep the customer traffic out of the tunnel.

Another area of further study concerns the mechanism for removing the contributions of the tunnels from the end-to-end loss rate and delays. In our initial implementation, we continually measure the roundtrip delay and loss between the monitor and each tunnel, and assign half of the measured loss and delay to each direction of the tunnel. This assumption of symmetric performance on the tunnels is clearly inadequate. We are working on techniques to provide us with better estimates of one-way delay and loss on the tunnels.

Finally, VMScope is an active probing mechanism that provides us with performance metrics across the provider backbone. It can be viewed as a data collection engine. We can build analysis and inferencing techniques on top of this data collection engine that will assist in network troubleshooting and performance monitoring. As such, we are hopeful that VMScope will serve as the basis for future provider-based MVPN performance monitoring.

6. REFERENCES

[1] H. Burch and C. Chase. Monitoring link delays with one measurement host. *SIGMETRICS Performance Evaluation Review*, 33(3):10–17, 2005.

[2] S. Casner and S. Deering. First IETF internet audiocast. *Computer Communications Review*, 22(3), July 1992.

[3] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, / 2000.

[4] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, Mar. 2000.

[5] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236, Nov. 1997.

[6] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[7] E. Rosen, Y. Cai, and I. Wijnands. Multicast in MPLS/BGP IP VPNs. Internet draft, Dec. 2004.

[8] E. Rosen and Y. Rekhter. BGP/MPLS Virtual Private Networks (VPNs). RFC 4364, Feb. 2006.

[9] K. Sarac and K. Almeroth. Supporting multicast deployment efforts: A survey of tools for multicast monitoring, 2001.

[10] K. Sarac and K. Almeroth. Monitoring IP Multicast in the Internet: Recent advances and existing challenges. *IEEE Communications Magazine*, 43(10):85–91, Oct. 2005.