Passive Aggressive Measurement with MGRP

Pavlos Papageorge*
Google
pavlos@google.com

Justin McCann
University of Maryland
jmccann@cs.umd.edu

Michael Hicks
University of Maryland
mwh@cs.umd.edu

ABSTRACT

We present the *Measurement Manager Protocol* (MGRP), an in-kernel service that schedules and transmits probes on behalf of active measurement tools. Unlike prior measurement services, MGRP transparently piggybacks application packets inside the often significant amounts of empty padding contained in typical probes. Using MGRP thus combines the modularity, flexibility, and accuracy of standalone active measurement tools with the lower overhead of passive measurement techniques. Microbenchmark experiments show that the resulting bandwidth savings makes it possible to measure the network accurately, but faster and more aggressively than without piggybacking, and with few ill effects to piggybacked application or competing traffic. When using MGRP to schedule measurements on behalf of MediaNet, an overlay service that adaptively schedules media streams, we show MediaNet can achieve significantly higher streaming rates under the same network conditions.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Protocol architecture; C.2.3 [**Network Operations**]: Network Monitoring

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Probing, Passive, Active, Streaming, Available Bandwidth, Kernel Module, Transport Protocol, Piggybacking

1. INTRODUCTION

The popularity of video streaming sites, live streaming of concerts, sporting and political events, and person-to-person voice and video conferencing have demonstrated the need for streaming systems that adapt to changing network conditions. Modern video and VoIP systems include some form of automatic rate determination during startup, but once a rate is set, scaling to higher rates can be problematic—how can the application learn that more bandwidth is available?

This and other problems—such as optimal overlay construction, multi-path routing, and optimal service selection—can be addressed using network measurement techniques to infer and respond to current network conditions. Broadly speaking, current techniques either *passively* observe existing traffic, or *actively* inject probe packets to see how the network responds. Using passive measurement, an application may observe the rate at which its buffers fill or empty [10] and monitor the rates of packet and frame loss [2, 3]. Passive observations are particularly useful for signaling that a stream's rate should be reduced, but do little to suggest when its rate can be increased [2, 3].

On the other hand, available bandwidth can be inferred by actively probing the network; when sufficient bandwidth becomes available, a stream could be upgraded. Such probing could take several forms. In the simplest case, the application may periodically improve the quality of the video or audio stream, increasing its transmission rate in the hope that the additional traffic can be supported. This has the obvious drawback that if insufficient bandwidth is available, session quality is harmed rather than improved [11, 17]. Another approach is to use an active measurement tool, such as pathload [14] or pathchirp [23] to probe the available bandwidth, and switch to a higher rate only if sufficient bandwidth is available. The drawback here is that the tool's probes compete with the application for bandwidth, which could again harm quality. Finally, we could modify the application to shape its own data as a measurement tool would, probing the link periodically to see if there is more headroom [16]. The benefit is that measurement traffic and application traffic do not compete, because they are one and the same. But this approach is *ad hoc*: an application coded to use one active algorithm may not be easily adapted to use another algorithm, and conversely a single measurement tool cannot reuse packets from several applications.

In summary: (1) passive measurement is efficient, but inadequate to detect when network conditions improve; (2) active measurement can detect additional bandwidth, but may adversely affect application traffic, diminishing its benefits; and (3) existing combinations of active and passive measurements tend to be implemented in a manner that is neither modular nor reusable.

*Work completed while the author was a student at the University of Maryland. This research was supported in part by NSF grant CNS-0346989.

SIGCOMM'09, August 17–21, 2009, Barcelona, Spain.

This paper presents the *Measurement Manager Protocol* (MGRP), an in-kernel service for performing network measurement that aims to address these shortcomings. The main novelty of MGRP is that it permits measurement algorithms to be written *as if active*, but implemented *as if passive*; in doing so, it enables measurement algorithms to be more *aggressive* in their approach, without harming application performance. MGRP works by transparently piggybacking application data inside probes, which mostly consist of empty padding. Rather than send their probe packets directly (e.g., via UDP), active measurement tools use the MGRP Probe API to specify an entire *train* of probes (by specifying the number of probes, the size of each probe, the amount of padding, and the gap between probes). MGRP treats each probe as a *vessel* that can potentially carry useful payload; data packets bound for the same destination ride along in the probe and are extracted and delivered separately when the probes arrive.

By filling most or all probe padding with useful data, MGRP allows active algorithms to approach the low overhead of passive algorithms, while retaining the accuracy, flexibility, and broader applicability of active probing. Measurement can be performed more aggressively, without worry of negatively impacting application traffic. Indeed, MGRP addresses our motivating example just as effectively as customized approaches [16] that are application-specific, but in a more modular, reusable fashion, which scales naturally to more applications and measurement algorithms. For example, we have implemented a measurement overlay network that actively measures its virtual links; applications query the overlay to acquire up-to-date path conditions. By running over MGRP, overlay probes can piggyback *any* application traffic traversing the same virtual links, and *any* active algorithm added to the overlay will immediately receive the benefits of piggybacking.

We have written a loadable kernel module for the Linux 2.6.25 kernel that implements MGRP as a Layer 4 transport protocol, and modified TCP to contribute packets to it (Section 2).[1] We have also modified three measurement tools to use the Probe API, including pathload [14] and pathchirp [23] for available bandwidth estimation, and badabing [28] for loss estimation. We have implemented a measurement overlay network to use MGRP-aware pathload to measure available bandwidth, and show how a modified version of the MediaNet streaming media overlay network [11] solves the upscaling problem by using the overlay measurements (Section 3). Microbenchmarks and experiments with MediaNet (Sections 4 and 5) show that MGRP is efficient, fair, and timely—MGRP improves the quality of service available to application source traffic and cross traffic, and enables measurement algorithms to probe more aggressively and thus complete more quickly. MGRP is available for download at `http://www.cs.umd.edu/projects/MGRP/`.

2. MGRP: THE MEASUREMENT MANAGER PROTOCOL

This section describes the design, operation, and implementation of the Measurement Manager Protocol, and discusses the effects of MGRP piggybacking on applications and measurement tools.

[1]We focused on TCP because of its broad applicability and to ensure that piggybacking does not cause harmful effects to its timing; we believe the results are applicable to UDP-based streams as well, and could be implemented easily.

```
 1 /* 5 probes, 1000 bytes each with 1 ms gap */
 2 char buf[1000];
 3 int probe_size     = 1000;
 4 int probe_data     = 20;  /* probe header */
 5 int probe_gap_usec = 1000
 6 int probe_pad      = probe_size - probe_data;
 7
 8 /* pass information using ancillary data */
 9 struct iovec v = {iov_base=buf,iov_len=probe_data};
10 char anci[CMSG_SPACE(sizeof(struct mgrp_probe))];
11
12 struct msghdr msg = {
13  .msg_name = daddr, .msg_namelen = sizeof(daddr),
14  .msg_iov = &v, .msg_iovlen = 1,
15  .msg_control = anci, msg_controllen = sizeof(anci)
16 }
17 struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
18 struct mgrp_probe *probe = CMSG_DATA(cmsg);
19
20 int s = socket(AF_INET, SOCK_DGRAM, IPPROTO_MGRP);
21
22 /* pass first probe to MGRP: activate barrier */
23 probe->barrier = 1;
24 probe->pad = probe_pad;
25 probe->gap = 0;
26 probe->piggybacking_allowed = 1;
27 sendmsg(s, &msg, 0);
28
29 probe->gap = probe_gap_usec;
30
31 /* pass probes 2..4 to MGRP */
32 for (i = 2; i <= 4) {
33    sendmsg(s, &msg, 0)
34    /* no delay between calls */
35 }
36
37 /* last probe to MGRP: deactivate the barrier */
38 probe->barrier = 0;
39 sendmsg(s, &msg, 0);
40 /* MGRP sends packet train before it returns */
```

Figure 1: Pseudo code of a probe transaction in MGRP.

2.1 Probe transactions

Active measurement tools require little modification to use MGRP. Rather than schedule and send their own probes via normal transport (e.g., UDP), measurement tools instead open an MGRP socket and submit a *probe transaction*, which is then scheduled via MGRP.

An example probe transaction is shown in Figure 1. Each probe is specified as a separate call to `sendmsg`, where the provided byte buffer (lines 9,14) defines the probe's header, and ancillary data (10,15,18) defines other characteristics. The `sendmsg` call for the first probe (27) sets the ancillary flag `barrier` (23) which activates a *virtual barrier* inside MGRP that delimits the beginning of a probe transaction. As long as the virtual barrier is active, MGRP buffers all subsequent messages from `sendmsg` calls (32-35) for that socket. Ancillary data is also used to indicate how much padding to include in the packet beyond the provided buffer (24), the desired interval between the transmission of the previous probe and the current probe (25,29), and whether the padding can be used for piggybacking data packets (26). Unsetting the `barrier` flag (38) deactivates the virtual barrier so that the last `sendmsg` call (39) does not return until MGRP has transmitted the probes as specified, with the desired header, padding, and inter-probe gaps. The sender may gather statistics about the probe train transmission via a subsequent `ioctl` call (see Section 2.4).

At the destination, the tool's receiving component calls `recvmsg()` on an MGRP socket to receive the probes. Each received probe consists of the header as provided at the

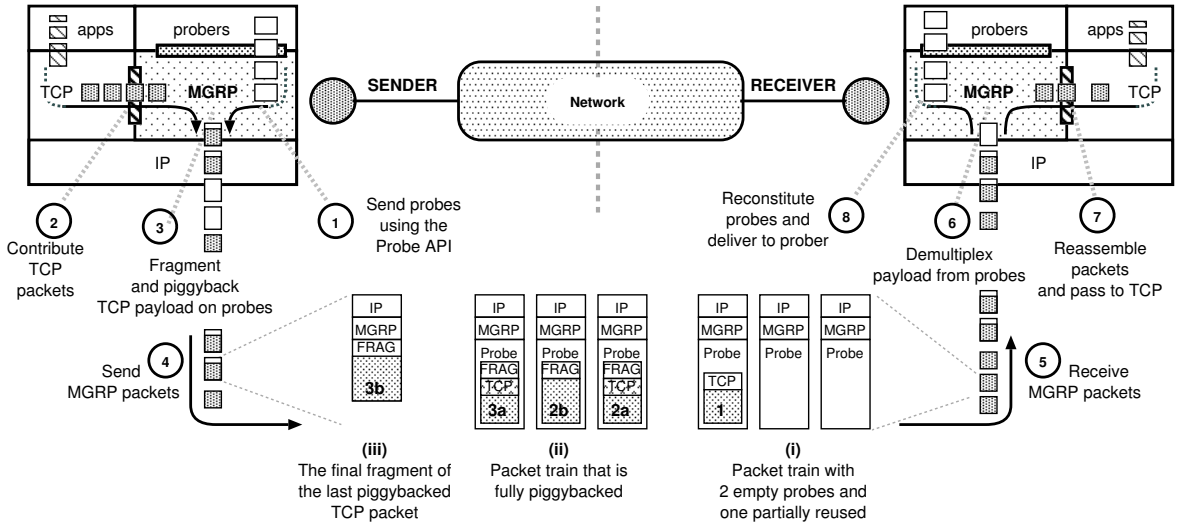


Figure 2: A step-by-step example to demonstrate MGRP operation. Measurement tools generate probes, which MGRP combines with TCP (or UDP) packets sent by applications. Depending on the timing, MGRP packets can contain (i) probes that are partially reused by full payload packets, (ii) probes that are fully reused by payload fragments, (iii) final payload fragments when MGRP has no more probes to send. The combined packets are demultiplexed at the receiver.

sender, along with any extra padding that was specified. Both the sender timestamp (implemented by MGRP) and the receiver timestamp (extracted from the kernel using the same mechanism as `SO_TIMESTAMP`) are provided as ancillary data. The tool uses this information to compute its results.

Though simple, this API is sufficiently flexible to implement the key building blocks—including single-packet probes, back-to-back packet pairs, and packet trains with arbitrary inter-packet spacing—of many existing measurement algorithms [5, 6, 12, 15]. We have adapted three measurement tools to use the Probe API: pathload [14] and pathchirp [23], which measure available bandwidth, and badabing [28], a tool that measures path loss. In all three cases, changes to the tools were minimal.

2.2 Basic Operation

MGRP's implementation of probe transactions is visualized in Figure 2. Once a probe transaction is fully submitted ①, MGRP attempts to fill the empty padding in each probe with rider packets drawn from application traffic bound for the same destination ②. For example, in our scenario from the Introduction, an application may be streaming media data to some destination D while an active measurement tool (such as our MGRP-adapted pathload) is used to probe the available bandwidth along the same path. Once any riders are piggybacked ③, MGRP stores the kernel timestamp in the MGRP header and hands the packet to the IP layer for transmission ④.

In our implementation, we modified the TCP stack to pass its outbound traffic to MGRP for potential piggybacking. Applications can use `setsockopt` to set which packets may be piggybacked. MGRP buffers potential riders for a fixed duration to increase the chances for successful piggybacking. If no riders are available, a probe is simply sent with empty padding (as in the first two probes of (i)). Any buffered application frames not piggybacked are sent normally.

Given a probe packet and a rider to piggyback, MGRP tries to fit the rider inside the probe's padding as follows. If the rider is smaller than the padding, piggybacking is straightforward (probe labeled 1 in (i)). If the rider is larger than the padding, MGRP fragments the rider into two chunks, piggybacking a padding-sized chunk in the probe and requeuing the remainder for further piggybacking. We use a simple, custom fragmentation/reassembly protocol similar to IP fragmentation to manage fragmented chunks.[2] In the figure we can see that probe packets in (ii) carry a single, fragmented TCP segment, whose chunks are labeled 2a and 2b. If MGRP cannot piggyback an entire rider before the buffer timeout, MGRP simply sends the chunk in its own MGRP packet, shown as (iii) in the figure. For simplicity we currently piggyback at most one rider per probe.

As probe packets are received at the destination ⑤, MGRP extracts any piggybacked riders from the probes ⑥. It then queues fragments for reassembly and immediately delivers complete TCP segments to the application ⑦. After extracting any piggybacked data, MGRP zeroes the padding and queues probes for delivery to the measurement socket ⑧. When the measurement tool retrieves the probe, it can retrieve the sender's and receiver's timestamps as ancillary data.

From this description we can see three main benefits to our kernel-based implementation. First, since all transmissions go through the kernel, MGRP can piggyback data from any number of applications on top of a probe train so long as it shares the train's destination. Second, applications require essentially no modification to donate their packets, while measurement tools require only modest changes. Finally, since we avoid crossing the user-kernel boundary for every probe, the inter-probe gaps can be generated with microsecond precision and low overhead using the high resolution

[2] Another approach we tried was to generate and add a new TCP header to each TCP chunk, which turned each chunk into a valid TCP segment. However, this approach turned a stream of MSS-sized segments into a stream of very small TCP segments (tinygrams) that interacted poorly with the ACK clocking and congestion window scaling mechanisms.

timers in the kernel. Other measurement services that perform a subset of MGRP services are also implemented as kernel modules, including MAD [27] and Periscope [9].

The main disadvantage of a kernel implementation is that it is harder to deploy, since it is OS-specific. We also considered implementing MGRP as a library that hijacked the various socket calls to implement Probe API extensions. This approach would be quicker to deploy, but lacks the first and third advantages of the kernel approach, making it less accurate and less efficient. Moreover, because piggybacking can only occur within the same application, more intrusive application modifications would be required (e.g., by managing a pathload-like service in a separate thread), making the approach less modular. One way to address this last problem is to redirect application data to a separate MGRP daemon, either via interprocess communication (IPC) through a hijacked socket interface, or via something like `/dev/tun` for IP tunnelling. While this approach could regain the advantage of modularity, it is still likely to be inefficient and less accurate. We spent significant effort attempting to implement the IPC-based approach but ultimately found it to be slow, cumbersome, and less accurate for want of high-resolution timers. Most importantly, by using high-resolution timers we ensure that TCP ACK clocking is maintained during normal transmission; earlier implementations with 10 ms timers performed poorly. Nevertheless, if deployability is the chief concern, a library-based implementation could work in some circumstances; our positive results concerning piggybacking suggest that such an implementation could be worthwhile.

2.3 Effects on Application Data

Piggybacking data packets within probes can reduce the number of packets and bytes sent across the network, compared to sending probes and application data separately. In the best case, this savings is enough to eliminate probe-induced congestion along the path, thus avoiding disruptive packet drops of source traffic, cross traffic, and measurement probes. Our experimental results show that this reduction in loss rates often leads to better application throughput, is more fair to cross traffic, and reduces the time required to complete measurements.

A potential drawback of piggybacking is that, while it can reduce the total number of lost packets, it can increase the chances that a lost packet contains application data. Observe that piggybacked application data is sent at the rate of the probes on which it piggybacks. Since measurement tools often send bursts of probes of high instantaneous bandwidth, these bursts are more likely to induce loss. If the probe burst rate is higher than the normal application data rate, then any lost packet is more likely to contain application data. For example, suppose that we send 10 probes per ms, while the application sends roughly 2 padding-sized packets per ms. With a 5-ms buffering timeout, we will buffer 10 application packets and can completely fill the probes' payloads. If we did not use MGRP at all, we would send 12 total packets—two application packets, and ten probes. While the chances of *some* lost packet may be greater in the second case, the chances of a lost *application* packet may actually be greater in the first case.

When a data packet loss occurs, MGRP can magnify the negative consequences of the loss because buffering increases the effective round trip time (RTT) for the application packets. While this additional latency is not necessarily a problem in and of itself, an increased RTT affects how TCP responds to packet losses, making it slower to increase the congestion window. So while longer buffering timeouts can reduce the total number of packet drops (by increasing piggybacking), they also delay TCP's response to packet losses (because buffering increases the RTT).

We address both problems by choosing a relatively small buffering delay, to balance the positive effects of increased piggybacking with the negative effects of the higher probability for packet loss. We have found that 5 or 10 ms is sufficient to buffer substantial amounts of data without harming TCP's responsiveness when dealing with typical wide-area RTTs of 40 ms or more [4].

In addition, we have found that carefully choosing *which* probes to use for piggybacking can positively impact performance. For example, pathload begins a measurement session by sending a burst of 100 packets back-to-back. If there is any congestion, the packets toward the end of this burst (and any packets that might follow it) are more likely to be dropped. If we selectively piggyback on only the first n probes of the burst, then if probes $m > n$ are dropped, no data packets will be dropped with them. The MGRP module could use external information to determine a suitable n automatically; e.g., a small TCP congestion window for a piggybacked data stream suggests the path is highly contended, increasing the likelihood of loss events.

2.4 Effects on Measurement Tools

By reducing the contention for a shared path, MGRP piggybacking allows a measurement tool to send traffic more aggressively, converge more quickly and/or provide more accurate results. Thus applications using these tools are better informed and can provide better quality of service.

However, when using MGRP, tools may need to be modified to account for piggybacked traffic. Normally, local traffic will be interspersed with the tool's probes, thereby affecting probe latency or queuing behavior. When piggybacking, MGRP removes some of this traffic from consideration, thus affecting a tool's calculation. Indeed, we have observed that without correction, available bandwidth tools will overestimate the available bandwidth when there is significant piggybacking.

To take the effects of piggybacking into account, tools can query MGRP via an `ioctl` call to learn piggybacking-relevant characteristics of the most recent probe transaction. In particular, a probe sender can query MGRP to learn: (a) how long it took the tool to queue probes to MGRP and how long it took MGRP to transmit the entire probe train, useful for verifying timing accuracy; (b) the number of probes and bytes that were piggybacked but would have competed with the probe train if MGRP were not enabled, useful for compensating for *hidden* data traffic in bandwidth estimates; and (c) the total number of probes that carried piggybacked data and the total number of bytes that were piggybacked across all probes, useful for gauging the actual overhead of the measurement tool.

To see how this information can be used to adjust an estimate, consider our MGRP-adapted pathload. Pathload estimates available bandwidth by trying to find the highest rate at which it can send data without observed congestion. To do this, pathload sends probe trains from source to destination in several rounds, where each train consists of 100 probes that are equally sized and equally spaced. After

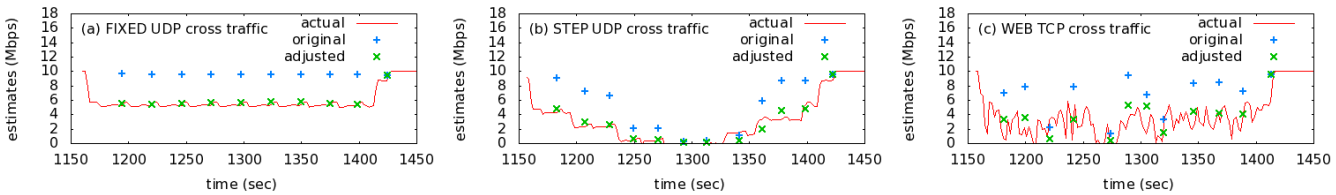


Figure 3: Adjusting pathload estimates for piggybacking (rate limited TCP source traffic with UDP and TCP cross traffic)

each round of probing, pathload uses the relative difference in the arrival times of probe packets to determine whether the network can support the current probe rate or whether there were queuing-induced delays. It uses this information to drive a kind of binary search toward the highest sustainable probing rate. Once it reaches this point, it computes a bandwidth estimate based on the rate of the final train. Note that if the network is highly saturated or the available bandwidth is highly variable, pathload may fail to converge to a fixed point and will simply time out.

Our modified pathload adjusts its estimates to account for piggybacking as follows. After each train the pathload sender uses the MGRP `ioctl` interface to retrieve the number of piggybacked application bytes that would have competed with the probes for the duration of the train, and sends this information to the receiver over pathload's control channel. When computing its estimate based on the last train, the receiver uses this information to compute the bandwidth consumed by the application's competing traffic, and subtracts it from the estimate. The approach works well. Figure 3 shows three time series plots that illustrate that the actual available bandwidth (red line) is estimated accurately by the adjusted pathload (green ×s), but overestimated by unadjusted pathload (blue +s).

3. MEDIANET AND THE MEASUREMENT OVERLAY

To demonstrate MGRP's benefit to applications, we used it to perform active measurements to improve the functionality of *MediaNet* [11], an overlay that delivers media streams. We first present some background on MediaNet, and then describe how we changed it to make use of a custom *Measurement Overlay* that actively measures the network while transparently making use of MediaNet's traffic.

3.1 MediaNet

MediaNet [11] is an overlay that aims to provide adaptive, user-specified quality-of-service guarantees for media streams. When requesting a streaming service, a participant specifies a series of preferred adaptations to use when available bandwidth is low. For example, the stream could be transcoded to use smaller frames or a lower-fidelity CODEC. Overlay node daemons, termed *local schedulers* (LS), apply any adaptations while forwarding traffic via TCP connections between nodes. The delivery path of a particular stream through the overlay, and any adaptations performed along that path, are determined by the MediaNet *global scheduler* (GS). The GS uses a simulated annealing algorithm that takes into account user preferences and current network conditions to configure the overlay to provide the highest possible utility to each user. In the best case, all users will receive their top preference, but if resources become scarce, some streams may be downgraded.

MediaNet uses purely passive measurement. Each LS periodically reports the rate at which it is sending application data to neighboring nodes. If congestion on a virtual link prevents an LS from sending at its preferred rate (beyond reasonable buffering), it alerts the GS of the rate it can actually achieve. The GS adjusts its view of the network accordingly, which may precipitate a reconfiguration, e.g., by rerouting a stream or by deploying an adaptation.

The problem with this passive approach is that under normal conditions it can only provide a lower bound on the bandwidth available on an in-use virtual link, and says nothing about the bandwidth available on a currently-unused link. Once MediaNet has lowered a stream's rate, the GS cannot tell when additional bandwidth becomes available, and so a user's utility may be kept lower than necessary once network conditions change (see Figure 17).[3]

3.2 Active Measurement Overlay

Remedying MediaNet's problems requires active measurement of both in-use links and currently-unused links to determine their available bandwidth. Then MediaNet can reroute paths and retract fidelity-reducing adaptations more effectively since it will have better, more up-to-date information about all its links.

With MGRP, such active measurements are made more efficient for in-use links by piggybacking on MediaNet traffic. To minimize modifications to MediaNet itself, we built a *Measurement Overlay* (MO) network that performs active measurements between its nodes. Currently, our implementation uses pathload to measure links' available bandwidth. By configuring the MO to mirror the structure of the MediaNet overlay (Figure 15 in Section 5), probe traffic from the MO can transparently piggyback MediaNet traffic when it is available. We modified MediaNet's GS to periodically query the MO to acquire per-link statistics for every link it needs to monitor; the LSes required no changes at all. These per-link active measurements complement the existing LS reports. Experimental results presented in Section 5 demonstrate these benefits: the GS can react quickly to reports of loss passively measured by the LSes, and safely increase the streaming rates when the MO reports higher available bandwidth. We also show that MGRP makes such available bandwidth estimates more efficient and more timely.

While we developed the MO for use with MediaNet, it demonstrates the modularity advantages afforded by MGRP. Any overlay/P2P network that has interest in measuring its active and inactive links can be set up alongside an MO, and thus receive timely and efficient estimates. Indeed, traffic from several overlay networks could be piggybacked on a single series of measurement probes, further improving

[3]Skype's algorithm for detecting additional bandwidth suffers a similar problem, and leaves sessions degraded longer than necessary [3].

the measurements' efficiency, without requiring any serious changes to the client overlay networks themselves. Moreover, decoupling the MO from its clients allows both to evolve independently. For example, as new measurement algorithms are developed, they can be incorporated into the MO and clients of the MO can immediately take advantage of them. Using the MGRP `ioctl` interface to assess the availability of data for piggybacking, the MO can be selective about which algorithms it uses to impose the least overhead. We envision an MGRP-enabled measurement substrate that provides modular, scalable, and efficient services to a wide variety of applications.

4. MICROBENCHMARK EXPERIMENTS

To understand the costs and benefits of MGRP, we first tested its performance with a series of microbenchmarks. The microbenchmarks provide us with a consistent data stream, cross traffic and number of measurement probes, allowing us to evaluate each MGRP parameter separately without having to factor in the effects of MediaNet rate changes. Using the insight gained from the microbenchmarks, we then implemented the Measurement Overlay and used it to improve the performance of MediaNet, presented in Section 5.

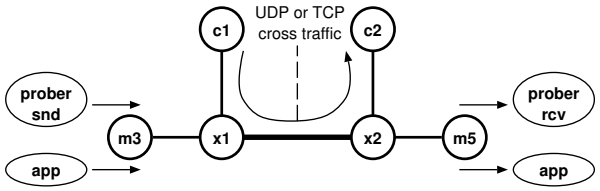
4.1 Experimental Setup

Our network topology and experimental parameters are shown in Figure 4. Source traffic is transmitted from m3 to m5, with varying amount of cross traffic sent from c1 to c2 across the bottleneck link x1-x2 which is shared with the source and probe traffic. We discuss each experimental variable in detail below. For each combination of variable values, we perform four 300-second runs, discarding any runs that did not complete due to experiment error. We measure the throughput of the source, cross, and probe traffic at the bottleneck link—i.e., after any losses have occurred—over one second intervals. When running pathload, we also measure how long it takes pathload to produce an estimate. Ideally, MGRP will improve the throughput of the source and cross traffic by reducing the amount of measurement traffic, and reduce the pathload completion times since we can now send probes more aggressively using mostly application traffic.

We ran the experiments on Emulab [34], using *pc3000* hosts connected via 100 Mbps Ethernet. We shape the x1-x2 bottleneck link using the FreeBSD Dummynet [24], limiting the link speed to 10Mbps (typical for a fiber-at-home user) and fixing the round-trip latency at 40 ms.[4] Dabek et al. [4] report that the median RTT between PlanetLab nodes is 76 ms, and between DNS servers worldwide the median RTT is 159 ms. Since higher RTTs lessen the penalty of MGRP's buffering delay, we present all results using a relatively low RTT of 40 ms. This is roughly the RTT between Washington, D.C. and Kansas City, MO.

Source traffic. We use *nuttcp* [7] to emulate a media stream by generating a TCP stream with a constant rate of 4 Mbps from $m3$ to $m5$. We chose 4 Mbps as the stream rate since this is the rate of high-definition streams for Apple TV and many other Internet video services [21]. We chose TCP as the transport protocol to emulate the use of Adobe's Real Time Messaging Protocol (RTMP) [22] and to also determine whether the buffering and time-shifting of TCP segments causes adverse effects (for example, due to changes to ACK clocking).

[4]Separate experiments with a wider range of RTTs (available in [18]) show that MGRP provides even greater benefits at higher RTTs. With no link delay, MGRP performs slightly worse than without (bulk transfer rates are 3-5% lower), since TCP recovery is delayed by the MGRP buffer.



Variable		Values
Traffic:	Source (m3–m5)	4 Mbps TCP
	Probe (m3–m5)	Pathload: `pFAST`, `pSLOW` Synthetic: pk1,pk2,pk3 (Tab. 1)
	Cross (c1–c2) (Fig. 5)	STEP (UDP) WEB (TCP)
MGRP		Off (`mgrpOFF`) On, 10 ms buffer (`mgrp10`)

Figure 4: Topology and experimental parameters

Probe Traffic. We considered two kinds of probe traffic: actual measurement traffic using two variants of pathload, and synthetic traffic meant to emulate a range of other tools and scenarios.

Pathload is the gold standard of available bandwidth tools, as it is both robust and fairly accurate [26, 30]. Pathload produces simple packet trains with uniform gaps, and uses more bandwidth than other tools (approximately 5–10% of the residual bandwidth in our experiments), which gave us an opportunity to test probe reuse in a non-trivial way.

As we show in Section 4.2, the original pathload is sometimes slow to return results. We found that between each train, the pathload implementation waits $\mathrm{RTT} + 9 * \mathrm{TX}_{train}$, where TX_{train} is the amount of time it takes to transmit one train. In an effort to increase the number of measurement rounds performed by pathload, we modified pathload to reduce this pause between trains to one RTT, which matches the description in the original pathload paper [14]. We call this more aggressive version `pFAST`, and the original implementation `pSLOW` in our experimental results. The main difference between the two is that `pFAST` sends probe trains more frequently (increasing the average probe rate) and therefore reaches an estimate more quickly.

One feature of pathload that is good for normal use, but poor for repeatability and for our MGRP evaluation, is that it responds to the conditions of the network by fluctuating the amount of probe traffic. This adaptive behavior makes it more difficult to distinguish between the effects of MGRP and the effects of pathload's implementation. Therefore, we also generate probes using our own tool called *pktrain* that sends custom packet trains that do not change size or rate based on network conditions. In doing so, we demonstrate that MGRP makes it practical to use more aggressive measurement tools with higher probe rates.

In our experiments we use three different packet trains, each with different size, timing, and transmission rates, as shown in Table 1. Note that all of the packet trains are rel-

	pk1	pk2	pk3
probe size (bytes)	300	500	300
probes/train	10	20	10
probe gap (usec)	200	1000	0
train gap (msec)	20	20	10
inst. throughput (Mbps)	12.0	4.0	line rate
avg. throughput (Mbps)	1.1	2.0	2.3

Table 1: Parameters for packet train experiments

atively short (10-20 packets). Each train type demonstrates different likely probing scenarios: *pk1* has a relatively high burst rate (12.0 Mbps) but low average rate (1.1 Mbps), and its ten-packet bursts approximate the sub-trains over which pathload calculates changes in one-way delay; *pk2* has longer 20-packet trains at lower rates (4.0 Mbps), but a higher overall rate (2.0 Mbps) demonstrating that more aggressive probing is feasible with MGRP; *pk3* is similar to capacity estimators such as pathrate, and transmits 10 packets back-to-back with an average probing rate of 2.3 Mbps.

Cross traffic. We generate two types of cross traffic between $c1$ and $c2$, shown in Figure 5: STEP and WEB.[5]

STEP cross traffic is stepwise UDP. We use tcpreplay [32] to replay a snapshot of a 150 Mbps trans-Pacific Internet link [31] at fixed-rate intervals lasting 45 seconds each (1, 3, 5, 6, 4, 2 Mbps). Using this UDP-based cross traffic, we can consider the effects of piggybacking on source traffic without having to factor in the cross traffic's response to induced congestion. Replaying a trace permits us to test against actual Internet packet size distributions and interleavings, but without cross-traffic congestion avoidance, and at rates that are stationary over a reasonable period of time [35].

WEB cross traffic is generated using the NTools [33] suite to generate up to ten concurrent TCP connections, which models web traffic using Poisson-distributed interarrival rates over 5-second intervals. Like the source, each TCP flow performs normal CUBIC congestion control. The flow size varies between 64 KB and 5 MB, and is weighted toward flows in the 64-200 KB range. The initial seed to the generator is fixed to provide consistent (though not identical) behavior between runs.

4.2 Results

The results of our experiments demonstrate that using MGRP leads to better application throughput, is fair to cross traffic, and reduces the time to complete pathload measurements. MGRP also makes it possible to measure at more aggressive rates without adversely affecting application traffic. On the other hand, highly-variable cross traffic (i.e., for the WEB experiment) in combination with bursty probe trains may increase source packet losses (as discussed in Section 2.3) unless we add more intelligence to MGRP to perform selective piggybacking as we discuss in Section 4.2.2.

4.2.1 STEP Experiment

The STEP experiment demonstrates all the potential benefits of MGRP. First, MGRP is able to piggyback significant amounts of application data, nearly eliminating probing overhead in almost all cases. As an example, Figure 6, plots a single run using pk2 traffic—notice that the bottom plot

[5]Additional results with other cross traffic patterns are available in [18].

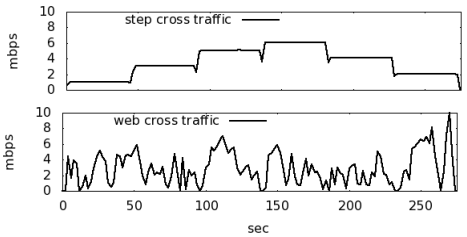


Figure 5: Cross traffic for Microbenchmarks

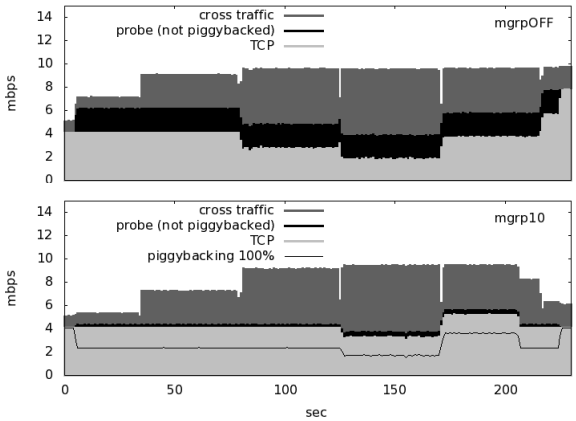


Figure 6: STEP: Timeseries plot with pk2 probes

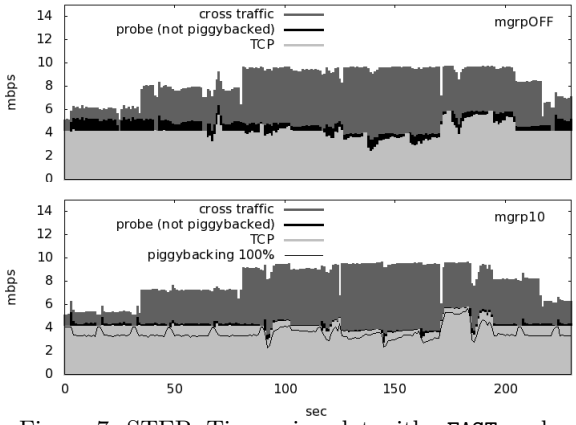


Figure 7: STEP: Timeseries plot with `pFAST` probes

has almost no black band, as piggybacking absorbs the overhead of probe traffic into the data traffic—and in Figure 7, showing the minor probing overhead when using MGRP with `pFAST`. Figure 8 shows the average per-second throughput of each kind of traffic for all runs. Probe-only traffic appears as a black bar, and riders ("source/probe (pbk)") contribute to the source throughput. The overhead of piggybacked probe headers is included with "probe (nopbk)," and is minuscule for most experiments with MGRP enabled.

Second, MGRP "smooths out" source traffic because it competes less with probe traffic. This effect can be seen in the time series plots mentioned above, and also in Figure 9, which shows the CDF of the source traffic throughput measured in one-second intervals, for different kinds of measurement traffic. Ideally, the CDF of our 4 Mbps source traffic would be a vertical line at 4 Mbps, indicating that the application was able to sustain its desired rate during the entire experiment. Any fluctuation below that rate (to the

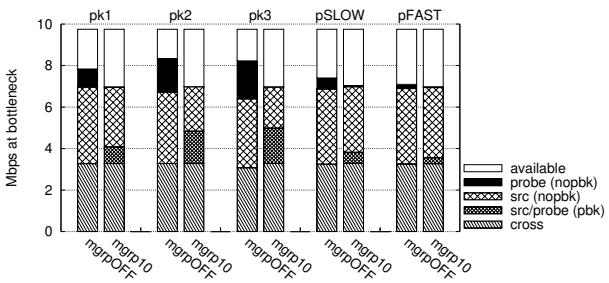


Figure 8: STEP: Average of per-second throughputs. Each pair of bars represents a different type of probe traffic.

left) is compensated for by a later return to higher throughputs in slow-start (to the right) when TCP is finally able to send queued application traffic. Without MGRP, the source rates for `pSLOW` and `pFAST` are comparable (not shown), but at higher probing rates used in packet trains pk1, pk2, and pk3, MGRP exhibits a significant improvement.

Third, MGRP imposes no adverse effect on UDP cross traffic, which sustains the requested rates with minimal losses. This is clear from Figure 8—the cross traffic portion is fixed for all runs.

Finally, MGRP allows pathload to complete its measurements more quickly, because there is less contention for the link. This can be seen in the CDF plots of `pSLOW` and `pFAST` completion times, shown in Figure 10. With MGRP, 50% of `pSLOW` measurements complete within 30.7 seconds, but without it, only 26% complete within that time and 48% fail to complete at all (we time out pathload rounds after 60 seconds). Thus MGRP leads to more measurement rounds that take less time to complete, without penalizing source or cross traffic.

4.2.2 WEB Experiment

As in other experiments, `mgrp10` greatly improves the time it takes for a measurement to complete (Figure 11). With `mgrp10`, `pFAST` measurements complete 20-25% faster at each of the 25th, 50th, and 75th percentiles, with even more significant improvements for `pSLOW`. Average pathload probing rates are also 20-25% higher with MGRP.

The WEB cross traffic is TCP, so we may expect that in the presence of probing traffic with high burst rates, the cross traffic suffers. However, because cross traffic is bursty, with idle periods, we find that its average throughput is largely unaffected, with or without MGRP, as seen in Figure 13. So the average per-second throughput is roughly the same with both MGRP on and off, with the synthetic packet trains having slightly better CDF distributions than `mgrpOFF` (not shown).

However, as can be seen in Figure 12, the source fares less well with MGRP and `pFAST` (right plot) when we look at the distribution of source throughput: over 40% of the time, the application is unable to sustain its target rate of 4 Mbps, but without MGRP the application pays a penalty only 20% of the time. With `pSLOW` (left plot), the MGRP penalty is less severe but still exists.

The reason for this is that pathload is *adaptive*. During its runs, if earlier trains experience few delays or losses, pathload increases the rate of subsequent trains. However, since the WEB cross traffic is highly variable (Figure 5), pathload

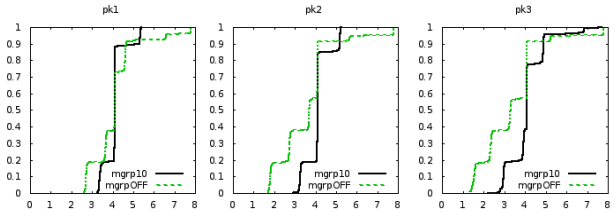


Figure 9: STEP: CDFs of source throughputs while running packet train probes pk1, pk2, pk3

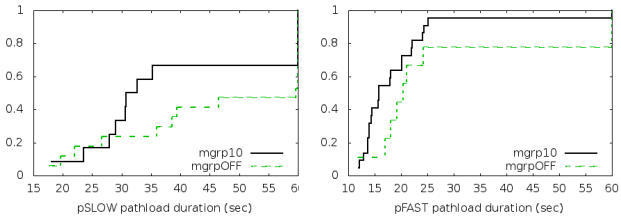


Figure 10: STEP: CDFs of pathload completion times.

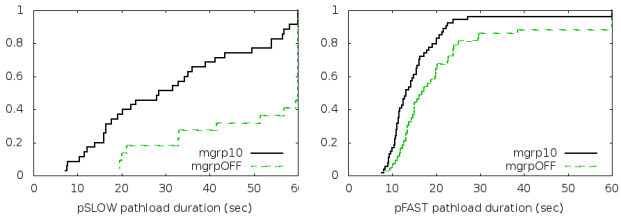


Figure 11: WEB: CDFs of pathload completion times.

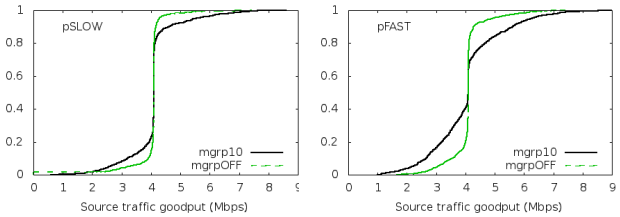


Figure 12: WEB: CDFs of source throughput with pathload.

is often too aggressive in increasing its rates, so that higher-rate trains are transmitted into a more congested network. As discussed in Section 2.3, while piggybacking can reduce the total number of lost packets, it can increase the chances that a lost packet contains application data. In particular, buffered source traffic is piggybacked onto high-rate, risky probe packets, increasing the chances for loss.

We can see this effect in the bottom plot of Figure 14, which displays a dot for each pathload probe. Each large grey rectangle shows a complete pathload measurement consisting of many probe rounds. Successful probes are light grey, dropped probes without riders (nopbk) are medium grey, and dropped probes with riders (pbk) are black. These black dots are application data losses, and cause TCP congestion avoidance to kick in, lowering throughput and increasing latency. As is apparent from the plot, source losses are highly correlated in particular probe trains (those at higher rates). They also tend to fall toward the end of probe trains (higher sequence numbers), when the bottleneck queue is already full of earlier probes and cross traffic.

We experimented with two approaches to avoid this problem: making the measurement tool piggybacking-aware, and using an MGRP regulator to limit the piggybacked probes.

In the first case, we modified pathload to selectively en-

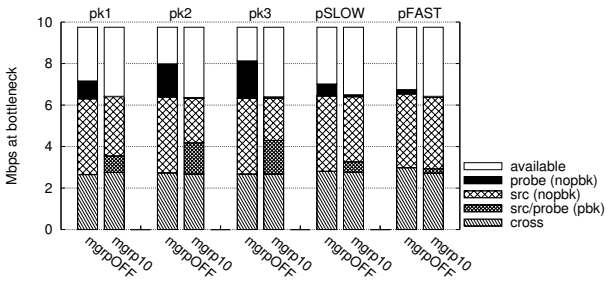


Figure 13: WEB: Average of per-second throughputs . Each pair of bars represents a different type of probe traffic.

able piggybacking on probes (via the Probe API) rather than blindly setting the piggybacking flag in the call to `sendmsg()` (Section 2.1). Our modified version (run in all experiments with MGRP on) disabled piggybacking during the initial "testing the waters" phase when pathload sends several high-rate bursts. We also considered having pathload avoid piggybacking on trains that are on the high side of its binary search [14], or selectively piggybacking only the first portion of those higher-rate trains, but implemented a more general solution instead.

In the second case, we added policy tuning knobs inside MGRP. One knob controls the maximum percentage of a train that MGRP will add riders to. By reducing the amount of the probe train that can carry riders, we trade off data loss in probe trains against the added congestion of competing traffic. Piggybacking on the first 50% of the train seems to improve measurement times while having a minor additional impact on source traffic. However, such a one-size-fits-all piggybacking percentage is probably not the best approach, since shorter probe trains (10-20 packets) will probably pay too high a penalty. The MGRP regulator could instead fix a number of probes it is willing to permit riders on (say, 50 consecutive probes in a transaction, rather than 50%), or calculate the number based on a TCP source's current congestion window.

We considered duplication-based mechanisms as well. Permitting data packets to transmit normally even when they have already been piggybacked on probes, removes the shared fate problem and gives each data packet two opportunities to reach the destination. Experiments showed such duplication does not reduce congestion and so does not perform as well. We considered duplicating data packets carried in a probe train, similar to Skype's behavior [1]. Since any unused padding is wasted network bandwidth, after a certain threshold we could simply repeat the data packets within the probe train and de-duplicate them at the MGRP receiver, but we have not implemented this.

5. MEDIANET EXPERIMENTS

To demonstrate how MediaNet can benefit from integration with MGRP we performed two groups of experiments considering: (a) the original MediaNet with only passive observations from the Local Schedulers, and (b) MediaNet integrated with a separate Measurement Overlay providing available bandwidth estimates to the Global Scheduler as described in Section 3, both with MGRP and without MGRP. We find that MediaNet's performance is improved by active measurement, and improves yet further when using MGRP.

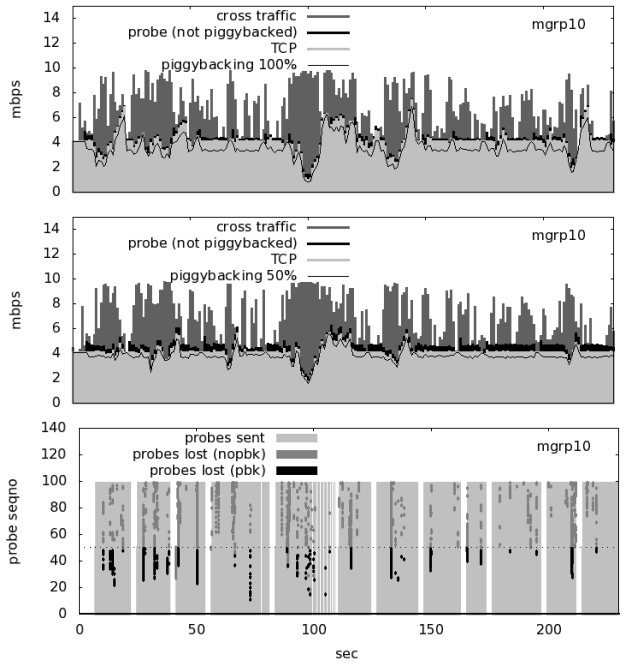


Figure 14: Effect of the piggybacking ratio on losses of the source traffic. The top figure shows 100% piggybacking and the middle shows 50% piggybacking. The bottom figure shows the losses inside the packet trains for the 50% case.

5.1 Experimental Setup

We use the network configuration shown in Figure 15. One local scheduler runs on each of the forwarder nodes (m3, m5) and the global scheduler runs on a separate node. In our enhanced version the Measurement Overlay also runs between m3 and m5, and the GS queries it for available bandwidth between those nodes.

Frame Type	Average Size (B)	Frequency (Hz)	Add'l BW (Kbps)
I	13500	2	216
P	7625	8	488
B	2850	20	456

Table 2: MPEG frame distribution

Source traffic. For source traffic, we loop three MPEG video streams. The MPEG streams consist of three frame types, I, P, and B (in order of importance). Lower-fidelity versions of a stream can be easily constructed by selectively dropping just the B frames, or the B and P frames. For our experiments, clients requested a particular stream and specified as adaptations either dropping just B frames (first choice), or dropping both P and B frames (second choice). For our particular source stream, the frequency and rates of these frames are shown in Table 2. Thus the full stream requires 1160 Kbps, while sending only I and P frames requires about 704 Kbps, and sending only I frames requires 216 Kbps. Since the experimental network permits only one path between the source and destination, all adaptation takes place by dropping frames, rather than rerouting. MediaNet drops frames as early as possible to avoid wasted bandwidth; in this case they are dropped at the sender.

experiment	runs	sec	Mbps	inc. over mgrpOFF	inc. over original	fps	inc. over mgrpOFF	inc. over original
mgrpOFF.pOFF	14	337	1.84			30.11		
mgrpOFF.pSLOW	22	336	1.96		6.29%	39.58		31.44%
mgrp10.pSLOW	32	336	2.05	**4.40%**	11.21%	43.42	**9.69%**	44.19%
mgrpOFF.pFAST	10	335	1.86		0.94%	39.10		29.87%
mgrp10.pFAST	22	336	2.28	**22.52%**	23.86%	52.08	**33.19%**	72.96%

Table 3: Increase of MPEG streaming rate with MGRP

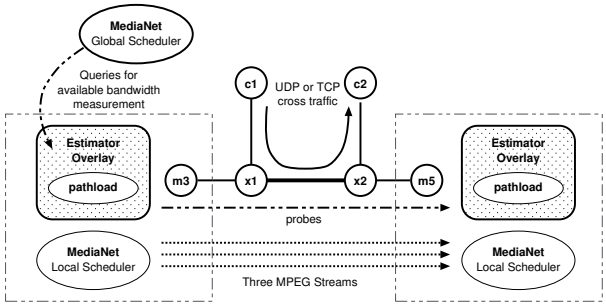


Figure 15: MediaNet experiment topology

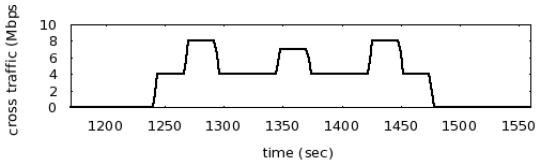


Figure 16: Cross traffic for MediaNet experiments

Cross traffic. Like our microbenchmarks, the MediaNet streams share the bottleneck link (x1x2) with cross traffic from c1 to c2. UDP cross traffic follows the PEAKS pattern shown in Figure 16, which adjusts the bandwidth in a stepwise fashion for 240 seconds using fixed-rate variable-duration intervals (4, 8, 4, 7, 4, 8, 4 Mbps). This creates multiple periods for MediaNet to increase its transmission rates, followed by periods of low available bandwidth.

5.2 Results

Table 3 summarizes the overall performance of MediaNet in the five different modes of operation: the original MediaNet without active measurement (`mgrpOFF.pOFF`, row 1), MediaNet with the original `pSLOW` pathload implementation (`mgrpOFF.pSLOW` and `mgrp10.pSLOW`, rows 2-3), and MediaNet using our more aggressive version of `pFAST` pathload (`mgrpOFF.pFAST` and `mgrp10.pFAST`, rows 4-5).

Results are collected from multiple runs (column 2) with the same average duration (column 3). To compute averages, we concatenate all runs of an experiment together and divide by the total number of seconds. The average total streaming rate for all three streams (Mbps, column 4) is the sum of the sizes of correctly decoded frames across all runs and divided by the total number of seconds. Column 5 presents the relative percentage improvement of using MGRP (rows 3 and 5) over not using it (rows 2 and 4, respectively). Column 6 presents the improvement of using active measurement at all (rows 2–5) over the original Medianet, which lacks active measurement (row 1). Notably, compared to active measurement without MGRP (`mgrpOFF`), MGRP improves the aggregate streaming rate by 4.4% when using `pSLOW` measurements and 22.5% for `pFAST`.

The average successful frame rate (column 7) is calculated by dividing the number of successfully decoded frames by the total number of seconds. Columns 8 and 9 report relative improvements, as above. It is obvious that MediaNet gains by using active measurement; even without MGRP there is a 31.44% increase in frame rate (39.58 fps vs. 30.11 fps for the original MediaNet). With MGRP, stream quality is even better—compared to `mgrpOFF`, `mgrp10` improves the frame rate by 9.69% for `pSLOW` (43.42 fps vs. 39.58 fps) and 33.19% for `pFAST` (52.08 fps vs. 39.10 fps)[6].

Note that without MGRP (rows 2 and 4), MediaNet is better off using the `pSLOW` measurements, even though they take longer to complete and sometimes do not converge. Without MGRP, the more aggressive `pFAST` scheme causes too many losses to data packets. With MGRP, more aggressive measurements are possible, leading directly to better application performance.

Medianet Plots. Figures 17, 18, and 19 each show a single run that illustrates the quality of service improvements shown in the table as a result of (1) incorporating active measurements using pathload (Figure 18), and additionally (2) using pathload with MGRP (Figure 19). All plots have time on the x-axis and depict 400 seconds of an experiment run that we have found to be representative of the many runs we considered.

The top of Figure 17 shows MediaNet's view of the available bandwidth on the network path between m3 and m5 in the presence of PEAKS traffic. The long horizontal blue line represents the amount of bandwidth that the MediaNet GS thinks is available. During reconfigurations, MediaNet ensures that the adapted media streaming rates combine to be less than the known available bandwidth. The gray shaded area indicates the actual available bandwidth, which is computed by subtracting the bottleneck traffic (the cross traffic and MediaNet streams) from the x1x2 link capacity (10 Mbps).

The MediaNet GS adjusts its available bandwidth estimate every time it receives a report. The original MediaNet relies solely on LS reports of frame decoding failures or excessive delay, and receives no information from active measurement tools.

The bottom of Figure 17 shows the per-second frame rate for the same run, with the y-axis representing the total number of frames per second decoded by the receiver for each of the three media streams. The first media stream is on the bottom in blue, with the 2nd and 3rd streams stacked on top of it in green and red (note the difference in y-axis from the top plot). After an initial setup period (also present in Figure 18 and most other runs), by time 1220 all three

[6]The increase in frame rate is higher than the increase in bandwidth because the additional frames are smaller in size, due to compression.

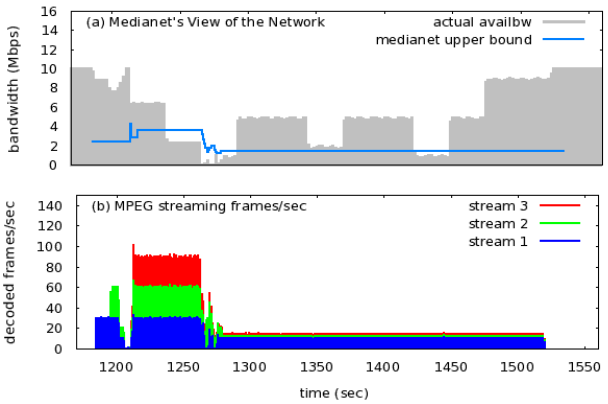


Figure 17: Original MediaNet (no active measurement)

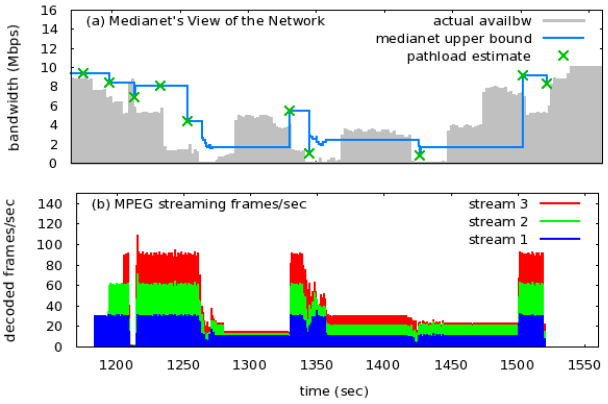


Figure 18: MediaNet with `pFAST`

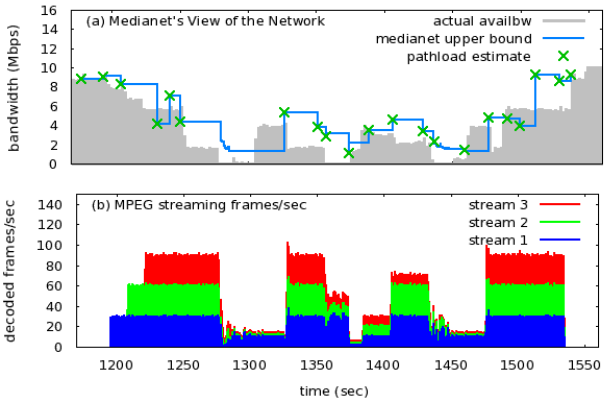


Figure 19: MediaNet with `pFAST` over MGRP

streams are received at their full rate. Once the cross traffic fills the available bandwidth around time 1270, MediaNet attempts to sustain the high rates, and finally drops stream 1 to medium rate (I and P frames) and streams 2 and 3 to low rate (I frames only). Since MediaNet is unaware of additional available bandwidth, it does not increase its rate.

In Figures 18 and 19 the Medianet GS is informed by regular estimates from `pFAST` pathload (green X's). With MGRP (Figure 19), MediaNet receives such estimates more often, and consequently follows the actual available bandwidth more closely and in a more timely manner. This enables MediaNet to take advantage of periods of increased bandwidth that are between congested periods, and bump all the streaming rates to high, which occurs in the periods at [1325,1350] and [1400,1425] in Figure 19. For the same time periods in the non-MGRP case (Figure 18) the streaming rates remain at medium. However, when the available bandwidth changes during a pathload run, pathload timeouts can occur with MGRP too, which is the reason why MediaNet did not upgrade the streams earlier in the period [1300,1350]. But it still outperforms the non-MGRP case.

6. RELATED WORK

MGRP is similar to a number of measurement services (Periscope [9], Scriptroute [29], pktd [8], precision probing [20], and MAD [27]) in that it provides a service to specify and schedule network measurements. MGRP differs from these in being the first fully implemented and usable service that fills empty probe padding with useful payload to reduce measurement overhead. We previously proposed the idea of seamlessly combining active probes with application data [19]; this earlier work had no kernel implementation and lacked real experimental evaluation.

MGRP's Probe API seems to be sufficient to implement the probing algorithms of many other measurement tools—including pathchar [6], pathrate [5], pathneck [12], and pathvar [15]. MGRP currently does not provide support for probes sent over ICMP, or for TTL-limited probes, but we believe we could add this support without difficulty.

Sidecar reuses existing TCP connections for measurement by reinjecting previously transmitted TCP segments to probe the network [25]. Doing so has the benefit that probes travel the same path as an actual TCP connection, so they can traverse NATs and firewalls without creating abuse reports. But Sidecar does not reduce congestion or make probe traffic less wasteful, as MGRP does. Moreover, since Sidecar probes are retransmitted TCP segments, potential probes are limited by the TCP state machine—to avoid impacting connection performance, Sidecar probes are transmitted only when the connection has been idle for at least 500 ms, which limits its utility in a streaming media scenario.

BitProbes is a service that leverages BitTorrent for large-scale Internet measurements, culling hundreds of thousands of target hosts from existing BitTorrent swarms, and measuring network characteristics to those targets using Sidecar and passive techniques [13]. While BitProbes uses BitTorrent traffic to find target hosts, MGRP reuses existing traffic to actively measure paths to hosts of interest.

Jain and Dovrolis [16] propose a streaming media overlay network called VDN (for *video distribution network*) that shapes the transmission of streaming media packets to resemble a pathload train, which it can use to infer available bandwidth. MGRP similarly allows application and measurement traffic to be scheduled together, and our MGRP-enabled MediaNet overlay (Section 3) performs a function similar to VDN (though it also measures alternate links on which no traffic is being sent). The key difference between VDN and our work is that with MGRP, applications and measurement tools can be kept separate. MGRP can piggyback *any* application traffic onto measurement probes with the same destination, and likewise, an application can easily take advantage of a range of tools. A newly developed tool simply uses the MGRP API to immediately benefit from the lower overhead that comes from piggybacking.

Passive measurement techniques, in which existing traffic is examined to infer characteristics of the network, have seen substantial research. MGRP is an optimization technique

for *active* measurement algorithms—by piggybacking actual application data on measurement probes, the overhead of active algorithms approaches that of passive algorithms.

7. CONCLUSIONS

We have presented MGRP, a kernel service for scheduling active measurement probe trains that can piggyback application data in what would otherwise be empty padding. The result is saved network bandwidth and reduced overall packet losses. We find that, compared to sending active probes interleaved with application data, piggybacking can improve application throughput, is fair to cross traffic, and reduces the time required to complete measurements. However, because MGRP causes probe packets and data packets to share the same fate, packets should be piggybacked selectively so as to reduce the chances that a lost packet is a data packet. MGRP allows tools and applications to be written separately and seamlessly combined. We think it can serve as foundation for new, low-overhead measurement services.

8. REFERENCES

[1] S. A. Baset and H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *INFOCOM*, 2006.

[2] Dario Bonfiglio et al. Tracking Down Skype Traffic. In *INFOCOM*, 2008.

[3] L. De Cicco, S. Mascolo, and V. Palmisano. Skype Video Responsiveness to Bandwidth Variations. In *NOSSDAV*, May 2008.

[4] Frank Dabek et al. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM*, 2004.

[5] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *INFOCOM*, 2001.

[6] Allen B. Downey. Using pathchar to estimate Internet link characteristics. In *SIGCOMM*, 1999.

[7] Bill Fink and Rob Scott. Nuttcp web site. `http://www.lcp.nrl.navy.mil/nuttcp/`.

[8] Jose Maria Gonzalez and Vern Paxson. pktd: A Packet Capture and Injection Daemon. In *PAM*, 2003.

[9] Khaled Harfoush, Azer Bestavros, and John Byers. PeriScope: An Active Probing API. In *PAM*, 2002.

[10] David Hassoun. Dynamic stream switching with Flash Media Server 3, April 2008. `http://www.adobe.com/devnet/flashmediaserver/articles/dynamic_stream_switching.html`.

[11] Michael Hicks, Adithya Nagarajan, and Robbert van Renesse. User-Specified Adaptive Scheduling in a Streaming Media Network. In *OPENARCH*, 2003.

[12] Ningning Hu et al. Locating Internet Bottlenecks: Algorithms, Measurements, and Implications. In *SIGCOMM*, 2004.

[13] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas E. Anderson. Leveraging bittorrent for end host measurements. In *PAM*, 2007.

[14] Manish Jain and Constantinos Dovrolis. Pathload: A Measurement Tool for End-to-End Available Bandwidth. In *PAM*, 2002.

[15] Manish Jain and Constantinos Dovrolis. End-to-end estimation of the available bandwidth variation range. *SIGMETRICS*, 2005.

[16] Manish Jain and Constantinos Dovrolis. Path Selection using Available Bandwidth Estimation in Overlay-based Video Streaming. In *IFIP Networking*, 2007.

[17] Aylin Kantarci, Nukhet Ozbek, and Turhan Tunali. Rate adaptive video streaming under lossy network conditions. *Signal Processing: Image Communication*, July 2004.

[18] Pavlos Papageorgiou. *The Measurement Manager: Modular and Efficient End-to-End Measurement Services*. PhD thesis, University of Maryland, 2008.

[19] Pavlos Papageorgiou and Michael Hicks. Merging Network Measurement with Data Transport. In *PAM*, 2005.

[20] Attila Pásztor and Darryl Veitch. A Precision Infrastructure for Active Probing. In *PAM*, 2001.

[21] David Pogue. For Purists, a Cut Above in Movies. Article in the New York Times, October 2008. `http://www.nytimes.com/2008/10/02/technology/personaltech/02pogue.html`.

[22] Robert Reinhardt. Beginner's guide to distributing Flash video. *AdobePress*, Sep. 2007. `http://www.adobepress.com/articles/article.asp?p=1014968`.

[23] Vinay Ribeiro et al. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM*, 2003.

[24] Luigi Rizzo. Dummynet web site. `http://info.iet.unipi.it/~luigi/ip_dummynet/`.

[25] Rob Sherwood and Neil Spring. Touring the Internet in a TCP Sidecar. In *IMC*, 2006.

[26] Alok Shriram et al. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *PAM*, 2005.

[27] Joel Sommers and Paul Barford. An Active Measurement System for Shared Environments. In *IMC*, 2007.

[28] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. Improving accuracy in end-to-end packet loss measurement. In *SIGCOMM*, 2005.

[29] Neil Spring, David Wetherall, and Thomas Anderson. Scriptroute: A facility for distributed Internet measurement. In *USITS*, 2003.

[30] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *IMC*, 2003.

[31] Samplepoint-F 150 Mbps trans-pacific trace (200803201500), March 2008. `http://mawi.wide.ad.jp/mawi/samplepoint-F/20080318/200803201500.html`.

[32] Aaron Turner. Tcpreplay tools. `http://tcpreplay.synfin.net/trac/`.

[33] Norbert Vegh. NTools traffic generator/analyzer and network emulator package. `http://norvegh.com/ntools/`.

[34] Brian White et al. An Integrated Experimental Environment for Distributed Systems and Networks. In *OSDI*, 2002.

[35] Yin Zhang and Nick Duffield. On the constancy of Internet path properties. In *IMW*, 2001.