Optimizing Data Transfer Nodes using Packet Pacing

Nathan Hanford
University of California
Davis, CA 95616
nhanford@ucdavis.edu

Brian Tierney
Energy Sciences Network
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
bltierney@lbl.gov

Dipak Ghosal
University of California
Davis, CA 95616
dghosal@ucdavis.edu

ABSTRACT

An important performance problem that we foresee with Data Transfer Nodes (DTNs) in the near future is a fast sending host over-running a slow receiving host, and packets getting dropped, leading to poor performance. Due to the design of the Transmission Control Protocol (TCP), if this occurs over a high-latency path, the performance impact can be quite dramatic. For example, for a 40 Gbps DTN sending to a 10 Gbps DTN over a 100 ms path, throughput can drop to less than 1 Gbps. Slow firewalls, under-buffered switches, or other devices in the network path that can not handle high speed flows also have this negative impact on throughput. In this paper we describe a simple tuning daemon at the sending host that detects flows when this is happening, and then tells the Linux kernel to throttle those flows to a rate that the network and receive host can handle. We also describe the results of several experiments to test the feasibility of such a solution, and in doing so are able to present some of the bounds of flow pacing at these line rates.

Categories and Subject Descriptors

C.2.1 [**Computer–Communication Networks**]: Network Architecture and Design; C.2.5 [**Computer–Communication Networks**]: Local and Wide-Area Networks—*internet*

1. INTRODUCTION

A Science DMZ is a portion of a network, built at or near a campus local network perimeter that is designed such that the equipment, configuration, and security policies are optimized for high-performance workflows and large data sets. The basic Science DMZ model [1] has been successfully implemented in numerous scenarios, including those involving astrophysics, photon science, high-energy physics, materials science, climate modeling, and genomics. These efforts have been notably recognized by the National Science Foundation, which has awarded multiple rounds of funding (as part of the CC-NIE [2], CC*IIE [3], and CC*DNI [4] Campus Cyberinfrastructure programs) to U.S. academic institutions to construct Science DMZ environments on their campuses to support research at scale.

For a classical Science DMZ, a network enclave is constructed using high-performance equipment (typically one or more switch/routers) at or near the institutional network perimeter. High performance servers, called Data Transfer Nodes (DTNs) are connected directly to these high performance routers. The DTNs handle all data ingest/export tasks, and need to be tuned for maximum network throughput.

Tuning a DTN for optimal performance can prove challenging. One problem is that most modern DTNs are fast hosts that are connected to the Internet at speeds of 10 Gbps or 40 Gbps (sometime 2x10 Gbps, 3x10 Gbps, and 4x10 Gbps as well). If these DTNs are sending data to a slower host or slower network, the receiving host or the router interfacing the slower network can drop a large number of packets, and performance can suffer dramatically due to TCP dynamics (as discussed in Section 2 below). Our tests show that TCP's congestion avoidance is not suitable to address the speed mismatch between the send host and the bottleneck device, especially with a 40 Gbps sender. The TCP congestion avoidance algorithm is designed to address congestion that occurs as a result of statistical multiplexing. In the case of bandwidth mismatch, what is required is to pace the sending rate to match the rate of the bottleneck device, and TCP's flow control is end-to-end. Therefore, a total solution to the problem requires a sender-side network flow control approach.

In this paper we show that using the Linux traffic control (`tc`) command to pace traffic on a per-subnet basis can dramatically improve overall DTN throughput. `tc` is a network layer traffic control function that can be used to shape, schedule, police and drop packets. It is implemented in the kernel and configurable. In this study, we consider the traffic shaping component which allows the rate of transmission at the egress to be controlled. Traffic shaping can also be used to smooth out bursts in traffic for better network behavior. In this paper we propose a `tc`-based flow control scheme to reduce the impact of the bottleneck device, and improve data transfer performance. The key idea is it to use `tc` based on pertinent information obtained from the Linux socket statistics (`ss`) tool.

We carried out experiments based on the most typical applications in order to test our hypothesis that such a packet-pacing management daemon would result in a significant improvement of the download time of a large data transfer. Furthermore, from our experiments, we determined a

INDIS 2015 November 15-20 2015, Austin, TX, USA

ACM ISBN 978-1-4503-4002-1/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2830318.2830322

recent upper-limit of packet pacing capabilities on contemporary commodity hardware. The results demonstrate that our daemon did significantly improve throughput and decrease retransmissions for long-running flows requiring large data transfers.

This paper is organized as follows: In Section 2, we discuss TCP issues and go into more details on DTN design. We also discuss related work on TCP pacing. In Section 3, we describe the various components of the `tc`based packet pacing tool. In Section 4, we discuss the experimental setup and in section 5 we discuss the results. Finally, in Section 6, we give a summary of the results and the future research directions.

2. MOTIVATION

Networks have become a critical piece of the scientific enterprise. Many disciplines now rely on networks to support the conduct of scientific research, above and beyond the ubiquitous use of email, web, and other commodity-like services. Scientific use of the network ranges from the collaborative use of multi-point high definition video conferencing systems to the global distribution of massive datasets. For example the Dark Energy Survey (DES) [5] is designed to probe the origin of the accelerating universe and help uncover the nature of dark energy, by measuring the 14-billion-year history of cosmic expansion with high precision. More than 120 scientists from 23 institutions in the United States have built an extremely sensitive 570-Megapixel digital camera, DECam, which is mounted on the Blanco 4-meter telescope at Cerro Tololo Inter-American Observatory high in the Chilean Andes. DES surveys a large swath of the southern sky out to vast distances (hence back in time), in order to provide new clues to the most fundamental of questions regarding the Big Bang and dark matter. DES gathers terabytes of data every night, and this will continue for 5 years. The National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign, which operates the data management pipeline for DES, receives 100,000 files each night. The bulk of the processing can be carried out in a data parallel fashion– each of the images within an exposure can be processed through the steps of detrending and calibration independently of the rest of the images. Their system is designed to capitalize on this data parallel capability by parceling the data out into independent jobs running in parallel. They use high-performance computing resources provided by the NSF's XSEDE (Extreme Science and Engineering Discovery Environment) project [6].

While the super computing and high performance computing facilities are connected to high-speed networks, when scientists attempt to run high performance applications over their institutional "general-purpose" networks, the result is often poor performance. With the increase of data set complexity and size, scientists often wait hours, days, or weeks for their data to arrive. The Science DMZ model was developed to specifically address these local area network issues, providing research institutions with a clear framework for improving their network environments to support data-intensive science. When developing the Science DMZ, several key principles provided the foundation to its design. First, these design patterns are optimized for science. This means the components of the system – including all the equipment, software and associated services – are configured specifically to support data-intensive science. Second, the model is designed to be scalable in its ability to serve institutions ranging from large experimental facilities to supercomputing sites to multi-disciplinary research universities to individual research groups or scientists. The model also scales to serve a growing number of users at those facilities with an increasing and varying amount of data over time.

2.1 TCP Performance

In addition to addressing the architectural issues that can hinder science productivity, the Science DMZ model was developed to combat the inherent performance limitations of Transmission Control Protocol (TCP) [7]; a broadly deployed protocol used for bulk data transfer and remote data access applications. While most scientific applications that need reliable data delivery use TCP-based tools for data movement, TCP's interpretation and reaction to network instability (e.g. the loss of data) can cause serious performance issues.

TCP is robust in many respects – in particular it has sophisticated capabilities for providing reliable data delivery in the face of packet loss, network outages, and network congestion. However, the very mechanisms that make TCP so survivable also make it perform poorly when network conditions are not ideal. In particular, TCP interprets packet loss as network congestion, and reduces its sending rate when loss is detected. For the many situations where the packet loss was not due to congestion, TCP backs off more than necessary, and can take a very long time to recover. In practice, even a tiny amount of packet loss is enough to dramatically reduce TCP performance, and thus increase the overall transfer time required. When applied to large tasks, this can mean the difference between a scientist completing a transfer in days rather than hours. Therefore, networks that support data-intensive science must strive to provide TCP-based applications with loss-free service if TCP-based applications are to perform well in the general case.

As described in ESnet's Science DMZ paper [1], an example of TCP's sensitivity was clearly demonstrated due to failing hardware. In 2012, ESnet had a bad 10 Gbps router line card that was dropping 1 out of $22,000$ packets, or .0046% of all traffic. Assuming the line card was working at peak efficiency, or $812,744$ regular sized frames per second[1], 37 packets were lost each second due to the loss rate. This resulted in an overall drop of throughput of 450 Kbps on the device itself, but reduced end-to-end TCP performance as demonstrated in Figure 1. This packet loss was not being reported by the router's internal error monitoring, and was only noticed using the `owamp` active packet loss monitoring tool, which is part of the perfSONAR Toolkit [8] [9].

Because TCP interprets the loss as *network congestion*, it reacts by reducing the overall sending rate. The sending rate then slowly (linearly) increases due to the dynamic behavior of the control algorithms, but can be further reduced at any point due to conditions on the network. This becomes more problematic as the distance between communicating hosts is large resulting in large round-trip times (RTTs). In such cases, the feedback (through ACKs) regarding bottlenecks in the network or the end-system takes longer to return. The relationship between latency, data loss, and network

[1]Performance Metrics. http://www.cisco.com/web/about/security/intelligence/network_performance_metrics.html.

capability was described by Mathis et al. as a mechanism to predict overall throughput [10]. The "Mathis Equation" states that maximum TCP throughput is at most:

$$\frac{\text{maximum segment size}}{\text{RTT}} \times \frac{1}{\sqrt{\text{packet loss rate}}} \quad (1)$$

Figure 1 shows the theoretical rate predicted by the Mathis Equation, along with the measured rate across ESnet. These tests are between 10 Gbps connected hosts configured to use 9 KByte ("Jumbo Frame") maximum transmission unit (MTU).

Note that the Mathis equation is for TCP Reno, and that there is no currently well accepted equivalent formula for TCP CUBIC [11] or H-TCP [12]. Figure 2 which plots the same data in log scale, clearly shows that shape of the curve is identical for Reno and H-TCP, so overall the formula still applies.

This example is indicative of current operational reality in science networks. TCP is used for the vast majority of high-performance science applications. Since TCP is so sensitive to loss, a science network must provide TCP with a loss-free environment, end-to-end. This requirement, in turn, drives a set of design decisions that are key components of the Science DMZ model.

2.2 The Data Transfer Node (DTN)

Systems used for wide area data transfers perform far better if they are purpose-built for and dedicated to this function. These systems, which we call data transfer nodes (DTNs), are typically PC-based Linux servers constructed with high-quality components and configured specifically for wide area data transfer. The DTN also has access to storage resources, whether it is a local high-speed disk subsystem, a connection to a local storage infrastructure such as a Storage Area Network (SAN), or the direct mount of a high-speed parallel file system such as Lustre[2] or GPFS[3]. The DTN runs the software tools used for high-speed data transfer to remote systems—typical software packages include GridFTP[4] [13] and its service-oriented front-end Globus Online[5].

DTNs are widely applicable in diverse science environments. DTNs typically have high-speed network interfaces such as a 10 Gbps or a 40 Gbps network interface controller (NIC). For example, DTNs are deployed to support a single beamline at Lawrence Berkeley National Laboratory's Advanced Light Source[6], and as a means of transferring data to and from a departmental cluster. At larger scale, sets of DTNs are deployed at supercomputer centers (for example at the United States Department of Energy's Argonne Leadership Computing Facility[7], The National Energy Research Scientific Computing Center[8], and Oak Ridge Leadership Computing Facility[9]) to facilitate high-performance transfer of data both within the centers and to remote sites. At even larger scales, large clusters of DTNs provide data service to the Large Hadron Collider collaborations—the Tier 1 centers deploy large numbers of data transfer nodes to support thousands of scientists. These are systems dedicated to the task of data transfer so that they provide reliable, high-performance service to science applications[10].

The set of applications that run on a DTN is typically limited to data transfer applications such as GridFTP. In particular, user-agent applications associated with general purpose computing and business productivity (e.g. email clients, document editors, media players) are not installed. This is for two reasons - first, the dedication of the DTN to data transfer applications produces more consistent behavior and avoids engineering tradeoffs that might be part of supporting a larger application set. Second, data transfer applications are relatively simple from a network security perspective, and this makes the appropriate security policy easier to apply.

Several campuses are currently deploying 40 Gbps DTNs, and 100 Gbps host NICs became available in June, 2015 [11], so production 100 Gbps DTNs are coming soon.

2.3 TCP Flow Pacing

Burstiness in network traffic can lead to packet loss which can cause reduced TCP throughput significantly in high-speed networks with large RTTs. Traffic pacing which attempts to space out packets, so that a group of packets clumped together are spaced out and do not result in packet losses in the network. TCP flow spacing has been studied extensively since the initial study in [14]. Much of this study has focused on ways to pace TCP and reduce burstiness. The benefits of TCP pacing on the network and the flow performance are not obvious and depend on many factors. Results in [15] show that in general TCP pacing can degrade TCP performance. On the other hand for networks with small buffers TCP pacing is required necessary to achieve high-link utilization [16]. Other than TCP pacing, packet drops in small buffer networks have been addressed using traffic conditioning [17] which rely on the global network-wide coordinated scheduling. These studies address buffer overflows that arise due to statistical multiplexing, and do not address the rate mismatches that arise in the case of DTN nodes addressed in this paper. Previous results have also shown that custom NIC hardware that does packet pacing can greatly improve performance [18].

3. TUNING DAEMON COMPONENTS

Our tuning daemon is based on a standard Linux TCP instrumentation available from the "socket statistic" command (`ss -it`) , and the "traffic control" packet pacing capabilities. These are described in more detail below.

3.1 Linux TCP instrumentation

Linux TCP sockets are well instrumented, and a wealth of information from the kernel made available in the user space via the `/proc` file system. The "socket statistic" command (`ss -it`) is a convenient way to get most of this diagnostic data, including the TCP round-trip time (RTT), the number of packet retransmits, the congestion window (`cwnd`), the window scale settings (`wscale`), which congestion avoidance

[2]Lustre. http//www.lustre.org/.
[3]GPFS. http://www.ibm.com/systems/software/gpfs/.
[4]GridFTP. http://www.globus.org/datagrid/gridftp.html.
[5]Globus Online. https://www.globusonline.org/.
[6]LBNL ALS. http://www-als.lbl.gov.
[7]ALCF. https://www.alcf.anl.gov.
[8]NERSC. http://www.nersc.gov.
[9]OLCF. http://www.olcf.ornl.gov/.
[10]LHCOPN. http://lhcopn.web.cern.ch/lhcopn/.
[11]Mellanox 100G NIC, http://www.mellanox.com/page/products_dyn?product_family=204&mtag=connectx_4_en_card

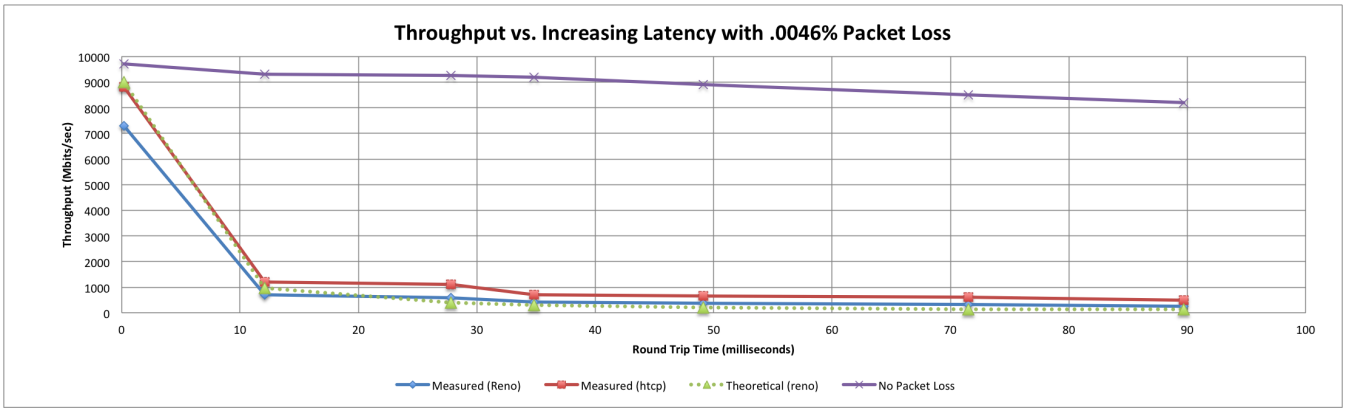

Figure 1: Throughput vs. latency for .0046% packet loss.

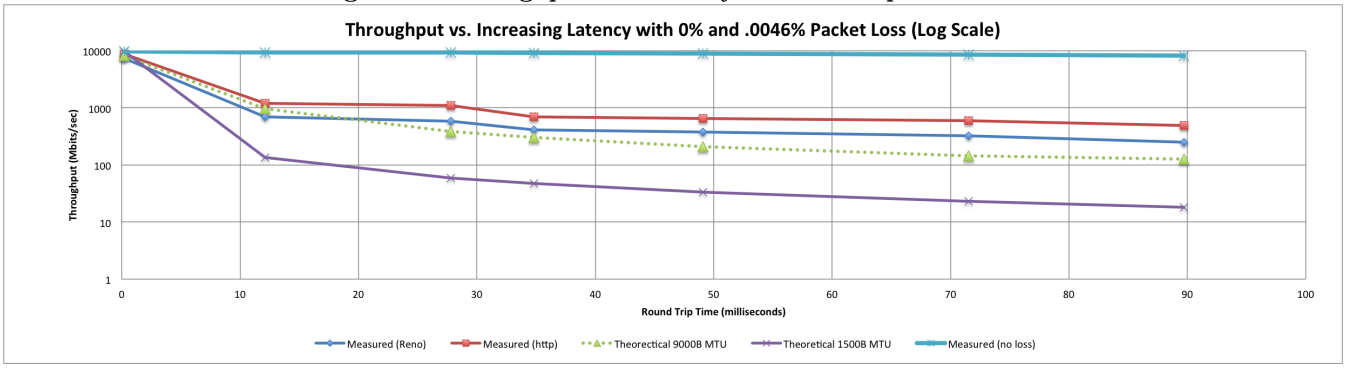

Figure 2: Throughput vs. latency: H-TCP vs reno.

algorithm is being used (e.g.: CUBIC, H-TCP, or Reno), and the current sending rate (`send`). While some information has been available from Linux for a while, the full set of information required by our tuning daemon is only available in kernel version 3.19 or higher. This means that depending on which OS distribution is installed, it may be necessary to install a newer kernel and corresponding version of the `iproute2` package, which includes `ss`. The interested Linux user-space system programmer can also access this information directly through the `netlink` interface and its derivatives, made available in various socket libraries.

Our tuning daemon uses the following information:

- We use a combination of the round-trip time, congestion window, and window scale to see if the host needs to be tuned. The daemon will recommend a set of host tuning commands based on this monitoring.
- We use a combination of retransmission, round-trip time, and congestion window data to see if traffic to a given host should be paced.

The details on how these data is used is described below.

3.2 Linux "traffic control" command

In Linux, traffic control is the term given to the entire packet queuing subsystem in a network or network device. Traffic control consists of several distinct operations. These include "classifying", a mechanism by which to identify packets and place them in individual flows or classes. "Policing" is a mechanism by which one limits the number of packets or bytes in a stream matching a particular classification. "Scheduling" is the decision-making process by which packets are ordered and re-ordered for transmission. "Shaping" is the process by which packets are delayed and transmitted to produce an even and predictable flow rate. Our tuning daemon is using the scheduling facility, and uses the "hierarchical token bucket" (`htb`) packet scheduler [19]. Details on how `htb` works is beyond the scope of this paper. We are using commands similar to the following:

```
#create a Hierarchical Token Bucket
/sbin/tc qdisc add dev ethX handle 1: root htb
#add a 'class' to our route queue with a rate
#    of 900Mbps
/sbin/tc class add dev ethX parent 1: classid 1:1
  htb rate 900mbit
#create a filter that restricts our tc queue
#    and class to a specific source subnet
/sbin/tc filter add dev ethX parent 1: protocol
  ip prio 1 u32 match ip dst W.X.Y.Z/32 flowid 1:1
```

3.3 Other components

Our daemon consists of a simple python script that calls `ss -it` every five seconds, and keeps track of active sockets. An ongoing summary for the entire data transfer is generated and stored in a SQLite [12] database. We are only interested in tuning long-lived "elephant" flows, so sampling once every five seconds is frequent enough to find these large flows.

[12] sqlite. https://www.sqlite.org/

3.4 Tuning Heuristics

Our daemon is based on the assumption that on most DTNs, the same users are retrieving files over long periods of time. We looked at GridFTP logs from the NERSC DTNs to confirm this. Focusing on the most heavy users of the DTN will lead to a bigger overall impact in terms of overall system performance.

We are not worried about every connection being tuned immediately, but rather we first establish a performance baseline, and then try to make it better. We collect data for ten transfers, and then start pacing traffic if our algorithm thinks pacing should help. We then watch ten more transfers, and if performance does not improve, we remove the pacing configuration for that path. To ensure our daemon learns about path or host upgrades, every two weeks we remove the pacing settings and start over.

We also try to figure out if there are multiple DTNs on the same subnet (i.e.: a DTN cluster), and pace traffic to the entire subnet, not just individual hosts, in order to reduce the processing overhead of unnecessarily pacing multiple flows to the same subnet separately.

3.4.1 Host Tuning

Our tuning daemon is designed for use on a DTN, which by definition should be well tuned, as described here: https://fasterdata.es.net/science-dmz/DTN/tuning/. However, for high latency, high bandwidth paths such as a 10 Gbps path between a DTN in the USA and an DTN in Europe or Asia, the default TCP buffer size it not big enough, and TCP performance will be limited by the congestion window size (`cwnd`). In this case, packet pacing will not help. However, using the RTT and `cwnd` information from the `ss` command, its possible to tell if the transfer is congestion window limited. This is very useful information to a DTN administrator, who may want to figure out which regular users of their service need to tune their hosts such that they are not limited by a congestion window that is too small. Our tuning agent logs this information in a clear, easy to search way to enable this sort of analysis. Our tuning daemon will detect this problem and generate a report on which remote endpoint hosts need tuning, and what that tuning should be. If the remote DTN is being managed by Globus, it would be easy for Globus to contact that user and suggest these tuning changes.

To determine if the path is congestion window limited, we compute the bandwidth delay product from the round-trip time reported by `ss`. We have no way to tell if the end-to-end path is 1 Gbps, 10 Gbps, or 40 Gbps, so we compute the optimal starting congestion window for a 10 Gbps path. However, this alone is not enough to consider a particular path as a candidate for pacing. Additionally, we monitor the maximum and average congestion window. Taking the difference, we can begin to determine which paths have discovered a larger throughput than they are experiencing in the steady-state.

We note that the round-trip time reported by `ss` is often more than double that reported by the ping command, so we use the shortest time seen by `ss` in this calculation, which is usually similar to the ping time. If the largest congestion window seen is less than 50% of this value, this is a good candidate for checking the remote host tuning settings. We also check which congestion avoidance algorithm is being used, and if `ss` reports TCP Reno, we suggest H-TCP instead.

Here is a sample tuning report:

```
Remote host 'hostname' may need to be tuned.
The host is CWND limited if it is connected at 10Gbps.
The host RTT is 100ms, and is running TCP Reno.
Suggest the following settings for /etc/sysctl.conf
   net.core.rmem_max = 268435456
   net.core.wmem_max = 268435456
   net.ipv4.tcp_rmem = 4096 87380 134217728
   net.ipv4.tcp_wmem = 4096 65536 134217728
   net.core.netdev_max_backlog = 250000
   net.ipv4.tcp_congestion_control = htcp
```

Our tuning daemon currently assumes both endpoints are Linux systems. There is not a good way to determine if the endpoint is some other operating system such as Microsoft Windows™ or Apple OS X™. More information on Linux host tuning is available at `http://fasterdata.es.net/host-tuning/linux/`.

4. EXPERIMENTAL ANALYSIS

At a high-level, our proof-of-concept design and deployment approach was split into three phases. First, we performed tests on two different closed-loop ESnet testbeds. Second, the tool was deployed as a monitoring system running solely in the user space on ESnet production DTNs. Third, the tool will be deployed into the ESnet production environment. The first stage involved using operating systems very similar to those found in the production environment. However, findings in the second stage prevented us from completing a full production deployment.

4.1 Testbed Experiments

To validate our results and ensure our method works across multiple host and network environments, we used two different test environments.

1. LAN test: Two Intel Haswell Xeon™ E3-1275 v3 3.5 GHz-based hosts with 40 Gbps and 10 Gbps NICs running Ubuntu Linux version 15.04, kernel version 3.19, connected by a IBM RackSwitch™ G8264 switch on a LAN. These machines are similar to our latest-generation perfSONAR [8] hosts. Therefore, they have less processing power and less memory than our latest-generation DTNs.

2. WAN Test: An Intel Haswell Dual-Xeon™ E5-2643 v3 3.4 GHz-based host with a 40 Gbps NICs running CentOS 7, Linux kernel version 3.10, to a AMD Opteron™ Processor 6140-based host, both connected to an Alcatel-Lucent SR7750 router, over a 50 ms RTT path and a dedicated 100 Gbps long-distance fiber-optic connection ("wave"). These machines are most similar to our latest-generation disk-based DTNs.

In both cases there was no other network traffic, so all packet loss was due to receive host issues. We also note that the results here are for a network path with a round trip time of .5 ms and 50 ms. If the path was longer, the performance without our proposed packet-pacing scheme would be much worse, as shown table 2.

For all results we ran `iperf3 -O 5 -t 30`, which runs 30 second single stream test, and ignores the first 5 seconds of the test to ensure TCP slow-start has completed. Each test

was run 10 times. Additionally, we ran experiments using GridFTP as a data application, throttling several simultaneous flows from 40 Gbps to 10 Gbps, in order to ensure the processor performance feasibility of pacing at these rates.

4.2 Production DTN Monitoring

After developing and deploying our traffic shaping tool on the ESnet testbeds, we modified the tool so that it would only run in a "monitor only" capacity. That is, it would not be able to actually shape any traffic, only monitor traffic from the user-space. We then deployed the monitoring tool on several DTNs in our production network. As with our testbed experiments, we monitored the same variables that the tool monitored when shaping traffic on the testbed, including Maximum Segment Size (MSS), Interface, RTT, Maximum and Average (`cwnd`), Retransmissions, and TCP implementation (e.g. H-TCP). This data was collected for each discrete flow over periods of several days at a time. The resulting dataset was examined to see if the heuristics employed on the testbed would be feasible in production. We will discuss the results of this monitoring in Section 5).

5. RESULTS

5.1 Testbed Results (Phase One)

Table 1 shows results with and without our packet pacing daemon for a 30 second test. Tests were done with 2 hosts (40 Gbps sender, 10 Gbps receiver) on a LAN with .5 ms RTT. Our testing showed that without pacing, we saw on average 367 packets retransmitted, for an average loss rate of .00001%. With pacing the packet loss rate was .00000001%.

We also ran similar tests over the WAN (also 40 Gbps sender, 10 Gbps receiver) to confirm the performance speedup results apply to other environments. Our testing showed that without pacing, we saw on average 826 packets retransmitted, for a loss rate of .000003% With pacing the packet loss rate was again only .00000001%.

Table 1: Test Results

No Pacing		
retransmissions	RTT	Throughput
367	.5 ms	9.4 Gbps
826	47 ms	7.1 Gbps
With Pacing		
retransmissions	RTT	Throughput
.3	.5 ms	9.8 Gbps
.2	47 ms	9.3 Gbps

We suspect the reason the impact of retransmissions on WAN performance is that the network devices on the WAN path had much more buffering, and that retransmissions came in bigger bursts. We believe this loss pattern means that the Mathis equation will not directly apply for this sort of network. This result also underscores the importance of the delay factor in the bandwidth delay product (BDP), and in turn, the Mathis equation. Our daemon will not pace flows with short RTTs as the effect on throughput is minimal, and therefore, it is not worth the processor overhead required of packet-pacing.

Perhaps one of the most critical points of this testbed experimentation is the processor overhead involved in pacing a high-speed flow. We determined, using `mpstat`, that pacing from 40 Gbps to 10 Gbps using GridFTP as an application with 1500 byte MTUs, created no significant increase in the processor resource utilization on either our WAN or LAN testbeds. The transfer was performed from `/dev/zero` on the sender to `/dev/null` on the receiver, to avoid a disk I/O bottleneck. In the worst case on the LAN testbed, the transfer used no more than 30% of a single core and 15% of a quad-core package. In the worst case on the WAN testbed, the transfer used no more than 30% of a single core and 10% of the 12 total cores on the system. However, there are upper-limits to pacing which we describe later.

5.2 Interpolated Results

Results for longer RTTs are shown in Table 2, and computed using the Mathis equation using the Switch TCP Throughput Calculator [20]. We note that since we are using the H-TCP and not the Reno congestion avoidance algorithm, these results will be low, but still indicative of poor performance, as shown in Figure 1.

Table 2: Results computed using Mathis equation

No Pacing		
retransmissions	RTT	Throughput
367	10 ms	8.3 Gbps
367	50 ms	1.6 Gbps
367	100 ms	.83 Gbps
With Pacing		
retransmissions	RTT	Throughput
.3	10 ms	9.8 Gbps
.3	50 ms	9.8 Gbps
.3	100 ms	8.3 Gbps

Based on these results, packet pacing using `tc` appears to be a very promising way to increase application performance. We note that there are limitations to what `tc` can do. For example, we have found that a 40 Gbps host can not reliably pace traffic to 35 Gbps due to processor limitations. The fastest `tc` can reliably pace on a modern host is around 20-34Gbps, depending on the speed of the CPU. This means that a very fast DTN capable of 39 Gbps sending to a slightly slower host that can only receive 36 Gbps may need to pace down to 30 Gbps or so to avoid heavy processor utilization.

We expect `tc`-based pacing to help in any environment where a slow device in the path is dropping packets. This includes firewalls, inline intrusion prevention systems, and so on. We hope to be able to test for this in the near future.

5.3 Results on ESnet DTNs (Phase Two)

We have begun testing our tuning daemon on ESnet's public anonymous GridFTP hosts (See: http://fasterdata.es.net/performance-testing/DTNs/). These hosts are available for anyone in the research and education community to use for performance testing, and typically do around 250 file transfers per day. As we mentioned earlier, rather than opt for a full deployment of the tuning capability of the daemon, we chose to first

use the daemon's internal monitoring component. In order to minimize resource utilization, our daemon collected data in intervals, polling `ss` on each interval. We chose a default interval of 5 s, in order to focus on data from elephant flows. Table 3 gives insight into some of the data the daemon collected. We were able to use hosts running on a slow (100 Mbps) connection at UC Davis in order to demonstrate how the daemon collects data from flows from an ESnet DTN located at Lawrence Berkeley Laboratory to a real university campus network. Because we had access to both of the endpoints in this experiment, we were even able to test a full-deployment of the daemon, using packet pacing, and restricting the pacing to only one subnet at UC Davis. However, as in many real-world tests, this was not a perfect experiment; H-TCP works very effectively at slow speeds and low latencies (5 ms in this case) so while we did not witness a significant gain in throughput, we did witness the expected results in our two key data categories: the average retransmissions per interval fell to 0, and the difference between the maximum and average `cwnd` fell by an order of magnitude.

Table 3: Results from our daemon's internal database

No Pacing	
Average retransmissions per interval	Difference between maximum and average `cwnd`
84.0	302168
With Pacing	
Average retransmissions per interval	Difference between maximum and average `cwnd`
0	31715

From these results we are comfortable making the claim that pacing flows does yield the effects that our daemon looks for when selecting candidates for pacing. That is, our daemon would not select this same flow for pacing again, given the fact that the retransmissions have fallen to 0. However, the converse of this claim is much more difficult to prove. Such a claim would be that our daemon is actually effective at selecting good candidates for flow pacing from a large set of unrelated flows, many of which could be suffering from a variety of different causes of retransmissions other than being congestion-window limited. Also, while the difference between the maximum and average `cwnd` is theoretically a good criteria for flow pacing, we have seen in practice that, even for flows on the same path, it can vary greatly, with or without correlation Figure 3 is an example of the difficulty in selecting candidates for pacing. All of the data points in the figure are separate flows from the same end-to-end connection. This intercontinental connection path has a known bandwidth of 10 Gbps, an RTT of 165 ms from the same DTN mentioned earlier, and an MSS of 1448 bytes. Since these flows were generated for testing purposes, they range in length from 20 s to 50 s, and are affected by a variety of different network conditions (unlike our closed-loop 100 Gbps testbed).

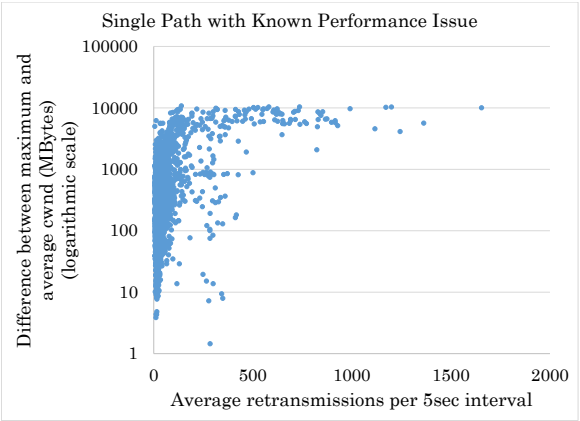

Figure 3: Key performance data for a high-throughput, high-latency path.

Figure 3 is just one example out of several different paths which have an interesting correlation between retransmissions and the difference between the maximum and average `cwnd`. So far, our daemon uses RTT, the difference between the maximum and average `cwnd`, and retransmissions as the criteria for determining good candidates for pacing. However, this is not enough. As we alluded to in Section 4, our lack of understanding of paths like these has delayed our full deployment of the packet pacing daemon (Phase Three). We will continue to employ other data sources as described in Section 6 to ensure our daemon does not generate false positives for pacing.

6. CONCLUSION AND FUTURE WORK

In Section 4, we described the type of data our pacing tool collects in its internal database. In Section 5, we described the subset of this data that our pacing tool uses to determine good candidates for pacing, but we noted that we are not yet able to claim that the tool can discern `cwnd`-limited flows from the thousands of flows originating from our production DTNs. Since our tuning daemon is designed to run on DTNs, and since GridFTP is commonly used on DTNs, it would make sense for our daemon to use the information in the GridFTP logs to help determine when to pace the traffic. Depending on what GridFTP server logging options are enabled, GridFTP can log things like transfer performance, number of parallel streams used, the file size, and the destination host. This is helpful information for the tuning daemon. We also want to do more testing on scalability limits for the number of `tc` route queue classes on a single host to ensure this will work for a heavily-used DTN. Furthermore, we are looking into `inet_diag` and `netlink` as possible callback-triggered data-sources.

From a practical perspective, we have concluded that the wealth of tools, including `ss` and `tc` made available in the `iproute2` package included with the Linux kernel provide a great resource for tuning end-systems concerned with a variety of network flows. The effective utilization of these tools should be of interest to systems engineers concerned with network performance. Furthermore, we have concluded that

it is indeed possible to use these tools to improve the performance of "elephant" flows in our application, particularly when the source of the flow is a 40 Gbps host.

7. ACKNOWLEDGMENTS

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research (ASCR), of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research was also funded by NSF grant CNS-1528087.

8. REFERENCES

[1] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. The science dmz: A network design pattern for data-intensive science. *Scientific Programming*, 22(2):173–185, 2014.

[2] National Science Foundation. Campus Cyberinfrastructure - Network Infrastructure and Engineering Program (CC-NIE). http://www.nsf.gov/pubs/2013/nsf13530/nsf13530.htm.

[3] National Science Foundation. Campus Cyberinfrastructure - Infrastructure, Innovation and Engineering Program (CC*IIE). http://www.nsf.gov/pubs/2014/nsf14521/nsf14521.htm.

[4] National Science Foundation. Campus Cyberinfrastructure - Data, Networking, and Innovation Program (CC*DNI). http://www.nsf.gov/pubs/2015/nsf15534/nsf15534.htm.

[5] Fermi National Laboratory and University of Chicago. The Dark Energy Survey. http://www.darkenergysurvey.org/index.shtml.

[6] NSF XSEDE. XSEDE - Extreme Science and Engineering Discovery Environment. https://www.xsede.org/home.

[7] J. Postel. Transmission Control Protocol. Request for Comments (Standard) 793, Internet Engineering Task Force, September 1981.

[8] perfSONAR Consortium. perfSONAR Technical Overview. http://www.perfsonar.net/overview.html, June 2009.

[9] Brian Tierney, Jeff Boote, Eric Boyd, Aaron Brown, Maxim Grigoriev, Joe Metzger, Martin Swany, Matt Zekauskas, and Jason Zurawski. perfSONAR: Instantiating a Global Network Measurement Framework. In *SOSP Workshop on Real Overlays and Distributed Systems (ROADS '09)*, Big Sky, Montana, USA, October 2009. ACM.

[10] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[11] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[12] Douglas Leith and Robert Shorten. H-tcp: Tcp for high-speed and long-distance networks. In *Proceedings of PFLDnet*, volume 2004, 2004.

[13] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, page 54, Washington, DC, USA, 2005. IEEE Computer Society.

[14] Lixia Zhang, Scott Shenker, and Daivd D Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. *ACM SIGCOMM Computer Communication Review*, 21(4):133–147, 1991.

[15] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of tcp pacing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1157–1165 vol.3, Mar 2000.

[16] Yan Cai, Y Sinan Hanay, and Tilman Wolf. A study of the impact of network traffic pacing from network and end-user perspectives. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–6. IEEE, 2011.

[17] Vijay Sivaraman, Hossam A ElGindy, David Moreland, and Diethelm Ostry. Packet pacing in short buffer optical packet switched networks. In *INFOCOM*, 2006.

[18] Takeshi Yoshino, Yutaka Sugawara, Katsushi Inagami, Junji Tamatsukuri, Mary Inaba, and Kei Hiraki. Performance optimization of tcp/ip over 10 gigabit ethernet by precise instrumentation. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–12. IEEE, 2008.

[19] Bert Hubert, Thomas Graf, Greg Maxwell, Remco van Mook, Martijn van Oosterhout, P Schroeder, Jasper Spaans, and Pedro Larroy. Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, page 213, 2002.

[20] SWITCH Foundation. Tcp throughput calculator. https://www.switch.ch/network/tools/tcp_throughput/.