NicPic: Scalable and Accurate End-Host Rate Limiting

Sivasankar Radhakrishnan*, Vimalkumar Jeyakumar+, Abdul Kabbani†,
George Porter*, Amin Vahdat†*

* University of California, San Diego   + Stanford University   † Google Inc.

{sivasankar, gmporter, vahdat}@cs.ucsd.edu   jvimal@stanford.edu   akabbani@google.com

Abstract

The degree of multiplexing in datacenters necessitates careful resource scheduling, and network bandwidth is no exception. Unfortunately, today we are left with little control to accurately schedule network traffic with low overhead on end-hosts. This paper presents NicPic, a system which enables accurate network traffic scheduling in a scalable fashion. The key insight in NicPic is to decouple the responsibility of state-management and packet scheduling between the CPU and the NIC, respectively. The CPU is only involved in classifying packets, enqueueing them in per-class queues maintained in host memory, and specifying rate limits for each traffic class. The NIC handles packet scheduling and transmission on a real-time basis. In this paper, we present the design of NicPic which offers a scalable solution for transmit scheduling in future high speed NICs.

1 Introduction

Today's trend towards consolidating servers in dense data centers necessitates careful resource management, which has been notably lacking for the network. It is not surprising that there have been several recent research proposals to manage and allocate network bandwidth to different services, applications, tenants and traffic flows in data centers. The bursty nature of data center traffic, together with high bandwidth and low latency networks has also revealed new challenges in congestion control [9]. This has led to new ideas for congestion control and burst mitigation in data centers to moderate various traffic sources and avoid overwhelming shallow buffers in commodity data center switches.

Many of these recent proposals can be realized on top of a simple substrate of *programmable rate limiters*. For example, EyeQ [6] and Gatekeeper [10] use rate limiters between pairs of communicating virtual machines to provide rate guarantees to tenant virtual machines in a data center. Proposals such as QCN [1], D^3 [12] use explicit feedback from the network to rate limit traffic sources and avoid bursts while apportioning bandwidth to different flows. These proposals require support for rate limiting a large number of flows or traffic classes on end-hosts. In virtualized data centers, this number can be in the thousands per server due to per-flow congestion control and per-source/destination VM traffic management.

Property	Hardware	Software
Scales to many classes	×	✓
Works at high link speeds	✓	×
Low CPU overhead	✓	×
Precise rate enforcement	✓	×

Table 1: Pros and cons of current hardware and software based approaches to rate limiting.

However, these new ideas have been hamstrung by the inability of NIC hardware to support scalable rate limiting and packet scheduling primitives. This has resulted in delegating scheduling functionality to software, which is unable to keep up with line rates, and diverts valuable CPU resources away from application processing. As networks get faster, this problem will get worse. We are left with a compromise between precise hardware rate limiters that are few in number [11, pg.3] and software rate limiters that are more scalable but suffer from high CPU overhead and burstiness (Table 1). While modern NICs support on the order of 100 queues [5, 8], to our knowledge, they only have a few (8–32) traffic classes with configurable rate limits.

In this work we present NicPic, a network stack architecture that combines the scalability of software rate limiters with the precision and low overhead of hardware rate limiters. The key insight in NicPic is to invert the current duties of the host and the NIC: we store packet queues in host memory, and have the CPU classify packets into those queues. The NIC handles packet scheduling and proactively pulls packets via DMA from host memory for transmission. In this model, the large amount of host memory provides scalability to support tens of thousands of classes, whereas the NIC enforces precise traffic schedules and rate limits in real-time. In the remainder of this paper, we identify the limitations of current operating system and NIC capabilities, and present a novel architecture that provides scalable rate limiting with low overhead.

2 Motivation

We begin by describing the applications that motivate NicPic, and then highlight some limitations of current host network stacks and the operating system–NIC interface that inform our design.

2.1 Target applications

Modern data centers house tens of thousands of servers, each with potentially dozens of CPU cores. The growing density of compute power drives a likewise increase in the number of flows that the NIC must service, schedule, transmit, and receive. Allocating network resources to flows is a critical challenge for enabling next-generation applications, driven in part by efforts to address the following challenges.

Bandwidth Management: Network bandwidth allocation for different services, applications and virtual machines in data centers relies on rate limiting or weighted bandwidth sharing [4, 6, 7, 11]. However, today's NIC support to handle fine-grained traffic scheduling is limited to a few (around 8–32) classes. With greater server consolidation and increasing number of cores per server, this situation is only going to get worse. As we place more virtual machines on any server, we need support for scheduling a larger number of individual flows and traffic classes. This necessitates a more scalable and flexible solution to control the transmit schedule on hosts.

Data center congestion control: Congestion control has typically been the responsibility of end hosts, as exemplified by TCP. Bursty correlated traffic at high link speeds, and small packet buffers in commodity switches can result in poor application performance [9]. High bandwidth and low latency in data center networks coupled with diverse workloads has led to new designs for congestion control. QCN [1], DCTCP [2], HULL [3], and D^3 [12] demonstrate how explicit feedback from the network can be used to moderate, rate limit or pace traffic sources and reduce congestion. The ability to precisely rate limit flows at a fine granularity enables new ways of performing congestion control and allows the network to operate at very low levels of internal buffering.

In summary, adopting new approaches to bandwidth management and congestion control requires fine grained control over transmission of packets to the network.

2.2 Limitations of Current Systems

Given the need for greater flexibility in scheduling packet transmissions, we look at relevant features in modern NICs and operating systems and discuss their limitations.

Figure 1 illustrates the packet transmit and scheduling pipeline in a state-of-the-art system with a modern NIC that supports multiple hardware queues. The operating system maintains a few transmit queues in host RAM from which packets are transferred to the NIC. When the operating system decides to transmit a packet, it sends a doorbell request to the NIC notifying it about the packet and the NIC Tx ring buffer to be used. The NIC fetches the packet descriptor from host RAM using DMA to its internal memory. Then the NIC looks up the physical address of the packet data, DMA's the actual packet and stores it in its internal Tx ring buffer. The packets in different NIC Tx ring buffers are finally transmitted on the wire according to a schedule computed by the NIC's packet scheduler.

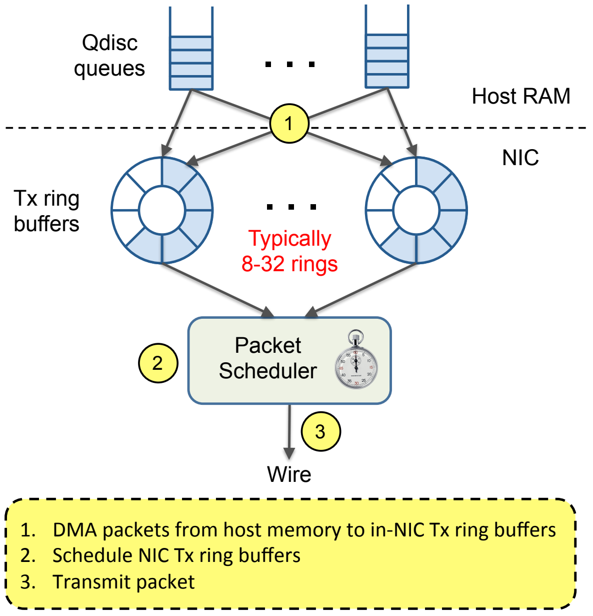

Figure 1: Current systems — "Pull and Schedule" model.

Single queue NICs do not need hardware scheduling and a basic "pull" model suffices. When NICs with multiqueue support for simple traffic prioritization or concurrent access from multiple CPU cores were introduced, a few hardware queues sufficed. The "pull and schedule" model which is a simple extension of the "pull" model works with such a limited number of queues. However, this model does not scale to support scheduling of traffic across large number of classes as necessitated by today's virtualized data center environments.

2.2.1 Hardware Rate Limiting

Modern NICs support multiple hardware transmit queues that are typically mapped to different CPU cores to enqueue and dequeue packets independently from each other, avoiding needless serialization through a single shared lock. Many NICs rely on simple inbuilt algorithms (such as round robin) to schedule packets from active hardware queues. Some NICs also support rate limiting in hardware. Each queue is configured with a separate rate limit [5], or there are a few (8–32) traffic classes with configurable rate limits and simple round robin among queues within a traffic class [8]. Such NICs provide a limited number of configurable rate limiters.

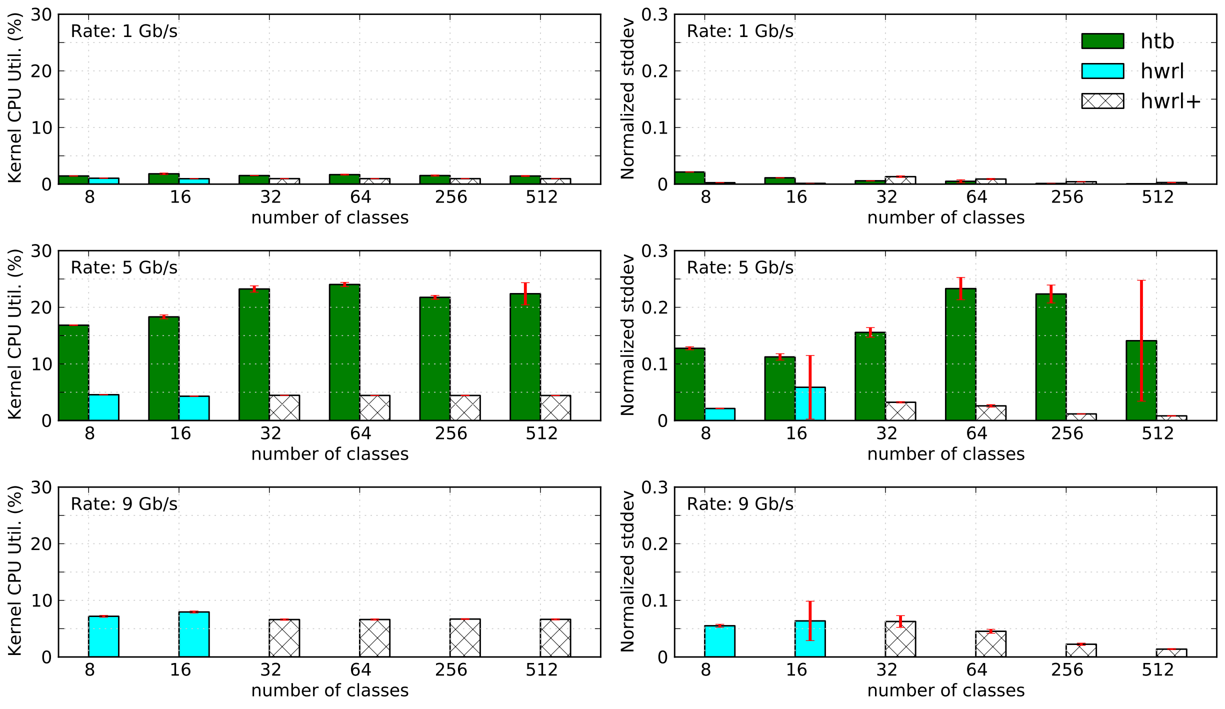

Figure 2: Comparison of kernel CPU overhead and accuracy of Linux's Hierarchical Token Bucket (htb) and Hardware Rate Limiting (hwrl, hwrl+). At low rates (1Gb/s or smaller), all approaches perform well, however at high rates (5Gb/s and 9Gb/s), hwrl performs best both in terms of CPU overhead and accuracy. Further, htb is unable to achieve more than 6.5Gb/s of aggregate throughput, and so it is not pictured at 9Gb/s.

2.2.2 Software Rate Limiting

Operating systems also provide support for controlling the transmit schedule. For example, Linux offers a configurable queueing discipline (Qdisc) layer for enforcing packet transmission policies. The Qdisc can be configured with many traffic classes from which packets are transmitted by the operating system. This software approach however has high CPU overhead and only supports coarse grained scheduling of traffic. The CPU overhead stems from lock contention and frequent interrupts used in computing and enforcing the schedule. The operating system typically batches up network processing and transfers packets to the NIC in bursts to reduce overheads, sacrificing precision.

The Qdisc ultimately dictates when packets are transferred from host RAM to the NIC hardware buffers, but the actual schedule on the wire is determined by the NIC. In addition, the operating system transfers packets to the NIC in batches, leveraging features such as TCP Segmentation Offloading (TSO). Depending on the batch sizes and how fast the operating system transfers packets, the NIC buffers might grow quite large. Once packets are in the NIC, the operating system loses control over packet schedules; packets may end up being transmitted at unpredictable times on the wire, frequently as large bursts (e.g., 64kB with 10Gb/s NICs) of back-to-back MTU sized packets transmitted at the full line rate.

In summary, current hardware approaches to rate limiting do not scale to many traffic classes, whereas software approaches have high CPU overhead and lack accuracy of rate enforcement.

2.3 Quantifying Accuracy and Overheads

To understand the CPU overheads and accuracy of rate limiting approaches, we benchmark a software rate limiter (Hierarchical Token Bucket, or htb) and a hardware rate limiter (hwrl) on an Intel 82599 NIC. The tests were conducted on a dual 4-core, 2-way hyperthreaded Intel Xeon E5520 2.27GHz server running Linux 3.6.6.

We use userspace UDP traffic generators to send 1500 byte packets. We compare htb and hwrl on two metrics—OS overhead and accuracy—for varying number of traffic classes. Each class is allocated an equal rate and we perform experiments with a total rate limit of 1Gb/s, 5Gb/s, and 9Gb/s. When the number of classes exceeds the available hardware rate limiters (16 in our setup), we assign classes to hardware rate limiters in a round robin fashion (shown as hwrl+). The OS overhead is the total fraction of CPU time spent in the kernel across all cores, which includes overheads in the network stack, rate limiter scheduling, and in servicing interrupts. To measure how well traffic is paced, we use a hardware packet sniffer at the receiver, which records packet timestamps with a 500 nanosecond precision. We compute the standard deviation of inter-packet arrival times for each class and normalize it to the class's ideal expected inter-packet arrival time. These metrics are plotted in Figure 2; the shaded bars indicate that many classes are mapped to one hardware rate limiter (hwrl+).

These experiments show that implementations of rate limiting in hardware are promising and deliver accurate rate limiting at low CPU overheads. However, they are constrained by their design, in that their scalability is limited by the number of queues they can maintain in on-NIC buffers. We also find that software-based rate limiters are impractical today at high line rates, and as rates continue to increase, will become increasingly infeasible. If we can leverage the hardware to use the large amount of host memory to store per-class queues, then we can both pace packets accurately while scaling to very large number of queues. In the following section, we present NicPic, which is our design for such a system.

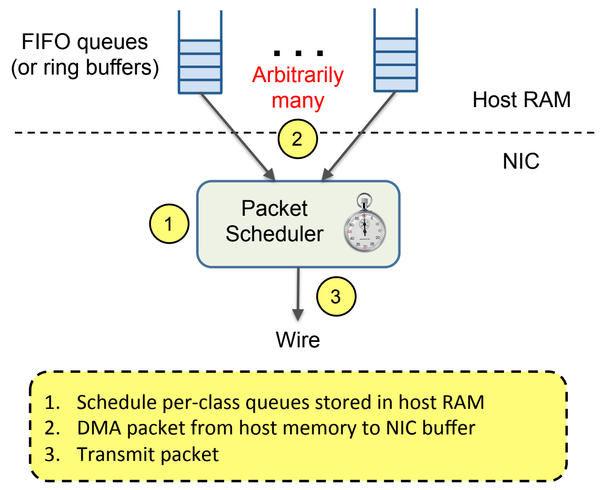

Figure 3: NicPic — "Schedule and Pull" model.

3 Design

NicPic proposes a design that delivers highly scalable and accurate rate limiting to lots of traffic classes with low CPU overhead. NicPic leverages the large amount of host memory (RAM) to store packets in per-class queues, rather than storing them on the NIC. Memory in the NIC is used to store metadata about those queues, which is quite small (e.g., 100s of kB for 10s of thousands of queues), achieving scalability. At a high level, the CPU classifies and enqueues packets in transmit queues, while the NIC computes a schedule that obeys the rate limits, pulls packets from queues in host memory and transmits them on to the wire. The NIC handles all real time per-packet operations and transmit scheduling of packets from different classes while satisfying their rate limits. This frees up the host CPU to batch network processing, which reduces overall CPU utilization. The NIC simply pulls packets from host memory using DMA when it is time to transmit them based on the schedule. This architecture is illustrated in Figure 3. We now describe these components in detail.

3.1 CPU functionality

As with current systems, the operating system or CPU retains control over traffic classification on the transmit side. Packets are classified by the operating system and enqueued in individual transmit queues. When the CPU enqueues a packet into a previously empty queue, it also flags the queue as active and notifies the NIC through a doorbell request that it should now consider this queue as well for dequeueing packets. The operating system or an application level process also configures rate limits for each traffic class. If kernel bypass APIs are used, then the operating system is responsible for granting access to separate transmit queues to different applications as appropriate.

3.2 NIC functionality

The NIC is responsible for all real time management of transmit queues. The NIC is dedicated to handling per packet operations in real time especially at higher link speeds. However it has limited hardware buffer resources. The NIC first computes the packet transmit schedule based on the rate limit. It figures out the next packet that should be transmitted, and only then pulls (or DMAs) the packet from the per-class queue in host memory to the internal NIC hardware buffer for transmission on to the wire. The NIC pulls packets from each queue in host memory in FIFO order, but services different FIFO queues based on the computed schedule.

On-demand scheduling: The NIC only schedules and pulls packets from host memory at the access link speed. So if the access link operates at 10Gb/s, even though a PCIe NIC might have much higher bandwidth between the host memory and the NIC buffers, packets are pulled from memory to the NIC only at 10Gb/s. The schedule is computed on demand and the NIC only computes the order in which a small number of future packets must be transmitted even if there are many more packets waiting in different per-class queues.

This late binding reduces the amount of NIC hardware buffers required for storing packets for transmission. More importantly, when a new traffic class or queue becomes active, the NIC can immediately consider the newly activated class for scheduling and service that queue, based on its rate limit. This avoids head-of-line blocking and unwarranted increases in latency between a new packet getting queued and when it can be transmitted by the NIC. In addition, this also allows the NIC to adapt quickly to changes in configuration. If the rate limits of some classes are reconfigured, the NIC can immediately compute schedules using the updated rate limits rather than still have several queued up packets using stale rate limits. This offloading of scheduling and real time work to the NIC is what enables NicPic to accurately enforce rate limits even at high link speeds.

Metadata handling: As mentioned earlier, the NIC needs to maintain some state about the active traffic classes or queues which would allow it to enforce the rate limits. This consists of global state for the packet scheduling algorithm itself, and some per-queue metadata. As a simple example, if we used token buckets for each queue to enforce rate limits, then the per-queue metadata includes tokens for each queue and the global state consists of a list of active classes which have enough tokens to transmit the packet at the head of the queue and another list of active classes waiting to receive tokens. The memory requirement for this metadata is fairly small, so supporting 10s of thousands of traffic classes requires only a few 100s of kB.

Segmentation Offload: With the new pull based approach, NicPic also modifies the NIC's TCP Segmentation Offload (TSO) feature to only pull MTU sized chunks of data from any queue in host memory at the time of transmission. This avoids long bursts of traffic from a single traffic class, and enables better interleaving and pacing of traffic. This is easily done by augmenting per-queue metadata with a *TSO-offset* field that indicates which portion of the packet at the head of the queue remains to be transmitted.

Since the NIC only maintains a small buffer of packets, and only pulls MTU-sized packets from memory at a time, it pipelines a sufficient number of DMA fetch requests to keep the link busy.

Rate Enforcement: NicPic is agnostic to the specific rate limiting algorithm used by the NIC. In addition to enforcing rate limits, we propose that the scheduling algorithm fall back gracefully to weighted sharing when the link is oversubscribed, i.e. the sum of configured rate limits of active traffic classes exceeds link capacity. One way to do this is to use a modified virtual time based weighted fair queueing scheduler which uses real time to enforce rate limits when the link is not oversubscribed and uses virtual time otherwise, to share bandwidth in the ratio of configured rate limits.

4 Conclusion

We presented NicPic, an approach to enable scalable and accurate rate limiting at end-hosts. The key idea is to reconsider packet handling duties between the NIC and CPU. The CPU should classify and enqueue packets in separate queues in host memory, possibly in batches to reduce overheads. The NIC should be responsible for computing the packet transmit schedule and pulling packets from per-class queues in host memory for transmission. This new separation of duties will enable scaling to many traffic classes while handling real time per-packet operations and enforcing precise rate limits.

References

[1] Alizadeh, M., Atikoglu, B., Kabbani, A., Lakshmikantha, A., Pan, R., Prabhakar, B., and Seaman, M. Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization. In *46th Annual Allerton Conference on Communication, Control, and Computing* (2008).

[2] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data Center TCP (DCTCP). In *Sigcomm* (2010).

[3] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *NSDI* (2012).

[4] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Towards Predictable Datacenter Networks. In *Sigcomm* (2011).

[5] Intel 82599 10GbE Controller. `http://www.intel.com/content/dam/doc/datasheet/82599-10-gbe-controller-datasheet.pdf`.

[6] Jeyakumar, V., Alizadeh, M., Mazières, D., Prabhakar, B., Kim, C., and Greenberg, A. EyeQ: Practical Network Performance Isolation at the Edge. In *NSDI* (2013).

[7] Lam, V. T., Radhakrishnan, S., Pan, R., Vahdat, A., and Varghese, G. NetShare and Stochastic NetShare: Predictable Bandwidth Allocation for Data Centers. *Sigcomm CCR* (June 2012).

[8] Mellanox Connect-X3. `http://www.mellanox.com/related-docs/prod_adapter_cards/ConnectX3_EN_Card.pdf`.

[9] Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D. G., Ganger, G. R., Gibson, G. A., and Seshan, S. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *USENIX FAST* (2008).

[10] Rodrigues, H., Santos, J. R., Turner, Y., Soares, P., and Guedes, D. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. In *WIOV* (2011).

[11] Shieh, A., Kandula, S., Greenberg, A., and Kim, C. Seawall: Performance Isolation for Cloud Datacenter Networks. In *HotCloud* (2010).

[12] Wilson, C., Ballani, H., Karagiannis, T., and Rowtron, A. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *Sigcomm* (2011).