Evaluating the Performance of Software NICs for 100-Gb/s Datacenter Traffic Control

Rob McGuinness
UC San Diego
jrmcguin@cs.ucsd.edu

George Porter
UC San Diego
gmporter@cs.ucsd.edu

ABSTRACT

Modern-day datacenters are constantly evolving and scaling to serve more users. Network traffic and flow control is a critical component to enable this trend, and there is a significant amount of new research in this space. Researchers need a platform for quickly implementing and experimenting with new traffic control mechanisms, while keeping up with increasing line rates. We examine the feasibility of implementing new methods of traffic control in software NICs. We analyze their ability to implement traffic control mechanisms at 40- and 100-Gb/s. We find that while they offer a platform for rapid prototyping at 40-Gb/s, they cannot (yet) support 100-Gb/s.

CCS CONCEPTS

• **Networks** → **Network adapters**; **Network simulations**; **Data center networks**; **Packet scheduling**; *Network protocol design*; *Network manageability*; • **General and reference** → General conference proceedings;

ACM Reference Format:
Rob McGuinness and George Porter. 2018. Evaluating the Performance of Software NICs for 100-Gb/s Datacenter Traffic Control. In *ANCS '18: Symposium on Architectures for Networking and Communications Systems, July 23–24, 2018, Ithaca, NY, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3230718.3230728

1 INTRODUCTION

The network interface card (NIC) is the interface between a host and the network. Traditionally the role and responsibilities of the NIC was clear, well-defined, and simple. The NIC transmitted packets generated by the operating system (OS) to the wire, and demultiplexed incoming packets from the wire to deliver them to the relevant input buffer. TCP's flow control and congestion control algorithms placed no particular requirements on when the NIC and OS actually transmitted packets on the wire, so long as no more than a certain number of packets were in flight at a time, as dictated by the receive and congestion windows. Recently, the adoption of datacenter networks, large-scale clusters, and rack-scale computers has fundamentally changed the interface between the server and the network. As a result, the NIC has become the "ground zero" of this reinvention, with commensurate changes in the requirements placed on it by application developers and network operators.

Several trends have driven the substantial change seen in endhost network stacks and network interfaces. End systems increasingly rely on virtualization to improve efficiency, either through virtual machines or via lightweight containers. The orchestration of traffic to and from these virtualized endpoints and the network requires network address translation, the implementation of access control lists (ACLs), and often custom forwarding rules, which are typically implemented in a virtual switch (vSwitch) abstraction. For performance reasons, the NIC has increasingly implemented this vSwitch functionality. A second trend is scaling across multi-core systems, which requires "steering" packets from the network directly to the core or hyperthread responsible for processing that flow.

A third trend is the adoption of radically new transport protocols, such as pFabric [2], NDP [11], Fastpass [23], and Ethernet TDMA [30]. Unlike TCP, these new transports commonly impose stringent requirements on exactly when packets need to be transmitted on the wire, and further often require fine-grained, per-flow rate limiting [12, 27]. Finally, datacenter operators and rack-scale computer designers have begun to explore new, advanced topologies. One such topology, based on expander graphs [14, 29], relies on non-shortest path forwarding and multi-hop indirection. Circuit-switched topologies rely on RF [7, 33] or optical [8, 17] devices to physically reconfigure their structure, which necessitates sending data at precisely the correct time [3, 18] (and rate) to match the physical configuration, potentially relying on multi-hop indirection as well [3, 20].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ANCS '18, July 23–24, 2018, Ithaca, NY, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5902-3/18/07…$15.00
https://doi.org/10.1145/3230718.3230728

Rapidly increasing link rates make these trends even more challenging to address. Networks running at 10- and 40-Gb/s have been deployed for years [25, 28], and 100-Gb/s NICs and switches are now commodity. At the 2018 Open-Compute Summit, a number of vendors announced 400-Gb/s networking gear to serve future datacenter workloads. While advancements in OS design have improved endhost performance, they are still hard to scale. This has led to the development of modular user-level, kernel-bypass frameworks called "software NICs" [10, 15, 22]. Software NICs are highly programmable software interfaces between virtualized endpoints (e.g., VMs or containers) and the network. While software NICs are primarily targeted for environments where more complex operations are required, such as network function virtualization (NFV), they are beginning to gain traction as a standard medium for endhost traffic control [26].

The pairing of endhost-backed flow control with the fine-grained capabilities of software NICs invites the natural extension that software NICs should implement these endhost-based flow control proposals, as the extensibility and ease of use of software NICs makes them extremely attractive for both rapid development and deployment in datacenter and rack-scale environments. In order for software NICs to be suitable to today's environments, their performance must be satisfactory for both current and future workloads.

In this paper, we investigate the feasibility of deploying complex endhost-based flow control and TDMA mechanisms on a representative modern software NIC, BESS [10]. As software NICs are deployed on faster links, first 40-Gb/s and ultimately 100-Gb/s, we seek to understand how they perform across a range of operational conditions. While we are evaluating BESS, its fundamental performance characteristics are largely due to DPDK [9], the underlying framework that it is built upon.

We create a number of experimental networking applications on the BESS framework and measure results up to and at 100-Gb/s speeds. We focus primarily on a limited number of targeted applications to determine performance baselines across core flow control concepts. We discuss our experience developing for the BESS architecture during development and some additional features that would be beneficial to recent flow control proposals.

We find that while BESS and DPDK provide mechanisms for designing and implementing wide range of networking applications purely in software, there are still some inherent limitations we observe that require NIC hardware support to solve. BESS provides great performance at speeds at 40-Gb/s and above, but fails to operate sufficiently at 100-Gb/s in the majority of the scenarios we tested.

2 BACKGROUND

In this section we discuss software network interface cards (sNICs) and their architectures, taking note of the author's original intentions of the problems they are intended to solve. Next we hypothesize how their software architecture can potentially provide a solution to current and future forms of high-speed network traffic control, particularly features that are often implemented via hardware rather than software. Finally, we introduce the core concepts of TDMA flow control, and how an sNIC may be leveraged for the requirements of TDMA schedules.

2.1 Software NICs (sNICs)

Software network interface cards are designed to supplement or reimplement functionality traditionally performed by NIC hardware, such as packet pacing, tenant isolation, and protocol offloading. Since these functions are implemented in software, they can be upgraded quickly, reducing bugs, enabling new features, and ensuring flexibility for future architectures and systems.

To provide high performance, sNICs layer directly on the hardware NIC by mapping portions of the NIC memory into the sNIC, rather than relying on an OS-provided device driver. Both BESS [10] and FlexNIC [15] directly claim to be software NICs, but we argue that NetBricks [22], a framework targeted specifically for network function virtualization (NFV) falls in this purview as well. Both BESS and NetBricks leverage DPDK [9], which provides drivers for mapping hardware NIC memory into userspace and a generalized library for interfacing with these drivers.

Of course, software may perform any amount of packet processing it wishes; NIC hardware does not limit the actions that endhosts can perform inside the network. But while the Linux TC subsystem provides a workable interface for many forms of traffic control, such implementations are typically too slow for current demands [16], leading to the trend of hardware offloading described above.

As modern datacenters are required to serve increasing amounts of demand each year [28], it is essential for sNICs to provide high performance alongside an easy-to-use framework for development. We argue that the long-term feasibility of sNICs is dependent on whether they can operate efficiently at high speeds or if iterations of specialized hardware will be necessary for future flow control needs.

For our evaluation, we use BESS as a representative case study for an sNIC. It supports recent versions of DPDK that interface with the 100-Gb/s NIC used in our tests. Our aim is to establish how well BESS achieves its goal for several different forms of traffic and flow control (which we refer to more generally as "flow control" or "traffic control") in a 100-Gb/s environment using several benchmarks. From this

we posit the strengths and weaknesses of an sNIC and its potential to support new forms of traffic control.

2.2 Endhost Flow Control

The advertised features of sNICs make them a suitable target for quickly changing implementations of network traffic control. At its core, traffic control aims to maximize network performance. It is important to quantify multiple distinct vectors of measurement in order to properly evaluate sNICs as a networking utility. We focus on a few of these general concepts that we believe are critical for future datacenter architectures.

2.2.1 Rate Limiting. Rate limiting is perhaps the most fundamental concept of flow control, with roots based in one of networking's earliest and most ubiquitous flow control proposal, TCP. The idea of rate limiting is simple: if the network cannot support the bandwidth that the host wishes to supply, simply make the host supply less bandwidth. This avoids packet loss, which can cause unnecessary retransmissions and increased latency.

Although hardware-based rate limiters have become common in recent high-speed NICs, they are unlikely to support per-flow limiting for thousands of flows due to limitations in on-NIC memory and processing power [24]. Recent work [26] has proposed a method to handle rate limiting in software with the help of an sNIC framework.

Many flow control proposals require rate limiting in order to avoid packet loss or restrict the bandwidth of low-priority flows. We list several such proposals in Table 1. It's clear that rate limiting is an essential component of flow control, and one that sNICs must support very well at scale in order to be a suitable alternative to hardware.

Table 1: Proposals that use rate limiting, TDMA, and/or indirection for flow control.

Proposal	Rate Limit	TDMA	Indirect
DCTCP [1]	✓		
EyeQ [12]	✓		
NDP [11]	✓		
TIMELY [21]	✓		
HUG [6]	✓		
Fastpass [23]	✓	✓	
Diamond [7]	✓	✓	✓
WaveCube [4]	✓	✓	✓
RotorNet [20]	✓	✓	✓
Eclipse [3]	✓	✓	✓
OSA [5]	✓	✓	✓
TDMA [30]		✓	
Helios [8]		✓	
ReacToR [17]		✓	
c-Through [31]		✓	
Solstice [18]		✓	

2.2.2 Packet Pacing. Because bandwidth is measured as data per second, the delta time that bandwidth is computed for can be of any duration. However, packets are always sent at the configured speed of the link. Even if the rate a flow is sending at is properly limited, the packets from that flow may be arriving in bursts large enough to cause packet loss. This can be mitigated with larger packet queues, which has the side effect of increasing overall latency.

In order to allow for small queues (and thus low latency) without overloading the network, some proposals require that rate limiters also pace packets. Perfect packet pacing is achieved by ensuring each packet of a flow is sent on the wire with interpacket gaps equal to the amount of time the packet would have theoretically required to send on a link with the same aggregate speed of the rate limited flow.

This is generally a feature relegated to hardware for two reasons. First, it requires software to precisely time when individual packets are given to a NIC, which requires a dedicated CPU core. Additionally, supplying a single packet in each PCI transaction to the NIC drastically reduces the potential throughput of the system compared to sending a group of packets as a single unit.

Some flow control proposals leverage packet pacing for additional performance. NDP [11] proposes a secondary queue for flow control administration that requires packet pacing in order to provide theoretical guarantees. We investigate what granularity of packet pacing an sNIC can perform to understand if it can potentially replace hardware solutions.

2.2.3 Flow Scheduling and TDMA. Flow scheduling can refer to a number of different specific approaches within the flow control space. One form of flow scheduling, Time-division Multiple Access (TDMA), is of recent interest due to its potential to be used in future optical and RF-based reconfigurable datacenter networks [5, 17, 30, 31]. In Table 1 we list some flow control proposals that leverage TDMA in their implementation. These proposals aim to provide a future-proof way to scale the speed of a datacenter at the cost of path availability, requiring a TDMA-style form of control in order to function efficiently.

TDMA flow scheduling operates on the core idea that flows cannot be transmitted at all times. TDMA schedules require sending bursts of packets to destinations during precise time windows. The exact reason for this restriction is dependent on the datacenter architecture and flow control system. For example, optical networks implement a circuit-switched abstraction, and so endpoints can only send data to a particular destination or set of destinations when the

circuit is established to those points. Fastpass [23] implements this circuit-switched abstraction on a packet-switched network, rather than on a physical circuit-switch.

We review two central concepts to TDMA scheduling: the *period* of the schedule, and its *duty cycle*. The period of the schedule is the duration over which a single set of flows may be sent, and includes any downtime delay from reconfiguration or other sources. We divide a period into an up-time during which packets are sent and a down-time when nothing can transit the circuit switch. The duty cycle of the schedule is simply the ratio of the up-time over the total period of the schedule, and represents the percentage of time that can be used to send data. The period and duty cycle of a dynamic TDMA schedule may change rapidly over time, such as in Fastpass.

TDMA flow scheduling can be difficult to efficiently implement on endhosts. Packets sent outside of the up-time can potentially cause increased queueing or even packet loss. This requires flows to start and stop transmitting as closely to the edges of the up-time as possible in order to maximize performance. Short periods or duty cycles make this a difficult task, but are theoretically beneficial by decreasing the scheduling latency of flows.

Little to no hardware support exists for TDMA schedules, and enforcing precise forms of packet transmission is intractable with the standard Linux networking stack [20, 30]. Software NICs may provide a potential solution to implementing this form of flow control at high speeds and realizing some TDMA-based flow control proposals which rely on simulations in order to compute their results [3, 18].

2.2.4 Multi-Hop Indirection. A unique form of flow control that has begun to appear in some systems is multi-hop indirection, a form of routing that has endpoints in the network forward a flow across multiple paths to its ultimate destination. Multi-hop indirection requires that components forward information back into the network along a transmission path, which reduces the potential outgoing throughput of the endhost or top-of-rack switch forwarding the flow.

Multi-hop indirection is rooted in the core ideas of Valiant load balancing (VLB), and has become a useful idea for flow control implementations in datacenter architecture proposals that do not always have a direct path between all endhost pairs. For example, the previously mentioned optical networking proposals often cannot implement a full crossbar using an optical switch. While an endhost could simply store the data until a direct connection is available, some proposals [5, 20] instead leverage underutilized outgoing links at currently connected hosts and employ multi-hop indirection to reduce latency. Yet other proposals [4, 29] may never have a direct connection between an arbitrary pair of hosts, and thus require indirection. We mark some proposals that use multi-hop indirection in Table 1.

Multi-hop indirection may be implemented via per-switch forwarding tables based on destination IP or other information, similar to how most tree-based datacenter networks typically operate. We examine a different method that provides forwarding information in each packet as an encapsulated GRE header, an idea taken from previous work in source-based datacenter routing [13]. Packet encapsulation is typically done in hardware, and most enterprise-level switches support using GRE headers to determine the output port to forward an incoming packet.

Software NICs provide a low-overhead method of prepending headers to packets, and are able to implement multi-hop indirection. Future datacenter architectures may significantly benefit from this if hardware solutions for multi-hop direction at 100-Gb/s speeds are unavailable or insufficient.

3 DESIGN

Software NICs are intended to be easily usable for a large range of networking applications, and must provide a consistent and extendable framework in order to fulfill this goal. They must accomplish this while also effectively leveraging an often unintuitive kernel-bypass framework in order to provide high performance for applications. BESS, the software NIC we analyze in this work, achieves this by having only a few core interfaces that users can combine and interact with to implement complex network functions. These interfaces leverage or even partially reimplement DPDK libraries to gain the performance benefits DPDK provides. BESS also exposes programmable RPC endpoints to interact with these interfaces in order to enable more dynamic systems.

We briefly describe the interfaces that BESS provides and how they can be leveraged by users to perform a variety of actions. We also discuss the potential limitations of this architecture and how it may be improved to provide even further flexibility. We then focus on the framework we created to run and gather results, and the modifications we made to the general purpose tools provided by BESS.

3.1 Core functionality

The main component of our sNIC consists of a C++ daemon that utilizes DPDK libraries. The daemon runs a gRPC (https://grpc.io) server that is used by a controller for setting up and configuring applications. A control client written in Python is provided to users in order to easily communicate with and execute tasks on a daemon. BESS applications consist of some combination of workers, schedulers, modules, and packets.

3.1.1 Components. A BESS worker is a logical thread pinned to a CPU core that executes a root scheduler object.

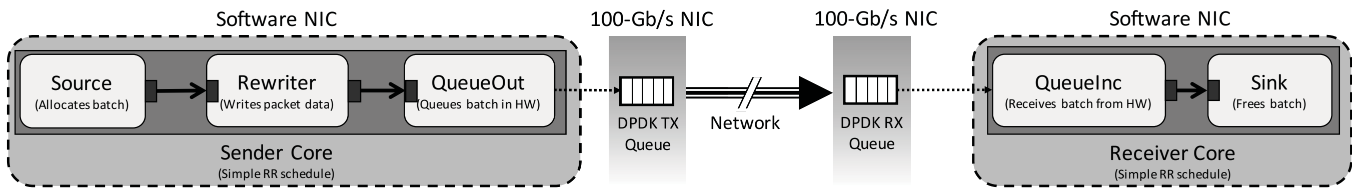


Figure 1: Simple sNIC sender and receiver. More cores may run concurrently with their own TX/RX queue.

Scheduler objects implement configurable scheduling policies, such as round robin or weighted fair queueing, and have one or more children attached to them, creating a tree. Each child may be either another scheduler object (a node), or a module that defines a task (a leaf). This is akin to how the Linux TC subsystem operates, and as such schedulers in BESS are also sometimes referred to as "traffic classes".

BESS modules may or may not have a task that is executed by a scheduler, but every module has some set of input and output gates that represent how packets flow through an application. An output gate may only connect to one input gate, though an input gate may have multiple output gates connected to it. A module that implements a task represents a starting point for some chain of modules that will handle batches of packets. When a module is run or receives some packets at an input gate, it will execute a function that may perform any operation it wishes over the batch, and then either stop all further processing of the batch or send the batch through an output gate to another module.

A range of basic modules are provided by the BESS codebase that implement several forms of common network functions, such as header encapsulation, field matching, and packet timestamping. Every module must supply a minimal set of RPC functions that allow the module to be created by the control client, and may supply further functions to allow users to inspect module state or change its configuration at runtime. Some modules provide wrappers around DPDK libraries, such as queue modules that may send or receive packets from a device queue on a DPDK-compatible NIC.

3.1.2 Simple Applications. All traffic in our results is generated by software NICs. We define a flow generated by an sNIC to mean a continuous stream of a single unique duplicated UDP packet. Every byte of each packet is written by the sNIC, and additional headers are written into the packet based on the experiment being run. We program the packet generation modules to run at their maximum speed, meaning there is always infinite demand for every flow. This ensures that we primarily evaluate the performance of the scheduling and packet processing components; we defer evaluation of timely traffic generation and application feedback for TDMA traffic patterns to future work. One or more flows may be sent via a single TX/RX queue, and each TX/RX queue pair is handled by a dedicated CPU core.

The simple sending and receiving applications we run for our microbenchmarks presented in Section 4.1 are shown in Figure 1. Source modules can be configured with different batch/burst sizes, which determines the number of packets that will be allocated and sent simultaneously to the pipeline, and ultimately over the network. The size of the data buffer given to the Rewriter module will determine the size of each packet in the batch. The statistics reported to the scheduler are used to determine sending and receiving rate.

3.2 Rate Limiting and Packet Pacing

BESS provides a rate limit scheduler that can attach to a single child task. We create one rate limit scheduler for each sending flow. Each worker uses a round robin scheduler as their root with rate limit schedulers as children. The rate limit scheduler uses an internal token bucket filter that is refilled at a user-programmed rate. After a batch of packets is sent by a flow, tokens are generated based on execution time. If a flow uses more tokens than are available, it is blocked for a period equivalent to the time it would have taken to produce those tokens.

This means that the rate limit scheduler attempts to perfectly pace the child when it is using bitrate as the variable for producing and consuming tokens. While the rate limit scheduler has a maximum burst parameter, this does not have any additional function when bitrate limiting is used. The true size of a burst for a flow is determined by the burst size that is input to the packet generation (Source) module, as this allocates a group of packets in a single execution loop of the scheduler. Because the rate limiter cannot block the flow in the midst of a single batch, large batch sizes at the Source module of a sending flow will result in a roughly equivalent burst of packets on the wire (n.b., the actual batch size sent is ultimately dependent on the NIC firmware).

3.3 TDMA Scheduling

TDMA support is not included in BESS. We implemented a custom TDMA module that runs on a dedicated CPU core with no other tasks. This ensures that the scheduler will not accidentally send packets at inappropriate times. While

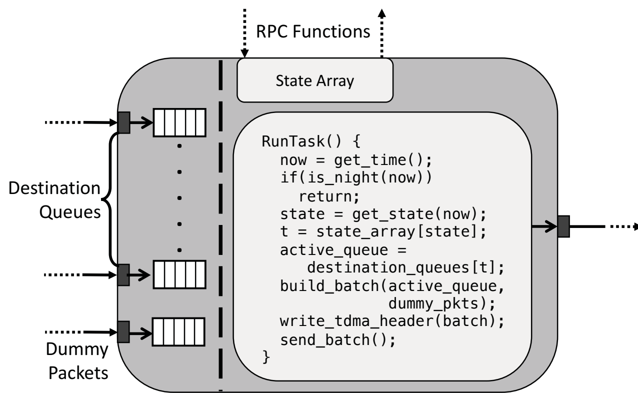


Figure 2: sNIC TDMA module. Destination queues may contain data from one or more flows. Dummy packets are used when little data is available to ensure the NIC transmits data packets promptly.

it would be expected to implement TDMA as a scheduling object, we have ultimately decided to implement TDMA as a monolithic module instead. The primary reasoning for this decision is twofold.

First, the module interface of BESS is clearly defined and allows for rapid development of new classes, and provides a direct way to create RPCs for these modules that can be used to inspect and modify the module's state. The scheduling interface does not permit heavy amounts of inspection or modification to overall state at runtime without significantly more programming effort, which made it difficult to use during development.

Further, the scheduling interface does not provide a way to the parent class to identify a specific node that has become blocked or unblocked, only that there was a child for which that event occurred. If a TDMA scheduler wishes to use rate limited flows or any flows that do not have infinite amounts of traffic, this then requires the scheduling object to manually iterate through all of its children to identify which child became blocked, resulting in an inconsistent and uncontrollable delay that scales with the number of flows the TDMA scheduler controls.

In general, we found the difficulty of developing new schedulers in BESS to be much higher than developing modules. Doing so requires a more in-depth knowledge of both the core daemon and control client, which may be disappointing for users that may be examining BESS as a general platform for future flow control systems. However, we have not examined the difficulty of creating a similar mechanism in the Linux TC subsystem, so we cannot comment on the difficulty of this task relative to BESS.

3.3.1 A Custom TDMA Module. The structure of our TDMA module is shown in Figure 2. The module is configured with the number of hosts in the network, and creates a destination queue for each. Flows provide packets to the appropriate destination queue via an equal number of input gates. The vertical line in the figure denotes a separation between the input gates which terminate the current task chain, and the output gate which executes a separate task on a dedicated core in order to ensure scheduling accuracy.

RPC functions are used to program the state array of the module, which contains the up/down pattern that the TDMA module task should use, and the appropriate destination queue to use for each up-time. Once the last state is executed, the module will loop back to the beginning of the state array. The state array also contains a base index time that is used to synchronize the up/down periods of TDMA modules on separate machines. We use PTP on a separate interface not controlled by software NICs to synchronize the host clocks of each machine.

There is an additional input gate for dummy packets, which are generated by a separate Source module and contain a basic Ethernet header with zeroed MAC addresses. This is necessary in order to ensure that the physical NIC transmits packets quickly even if there is only a small number of packets waiting in a destination's queue when up-time begins. If a batch cannot be completely filled with real packets, dummy packets are used instead. Because the MAC addresses are zeroed, a network switch will drop these packets immediately upon receiving them.

Even with this precaution, there are still times that the physical NIC hangs onto packets too long, causing packets to be transmitted during down-times. Additionally, the delay between the NIC receiving and transmitting its first batch can be high. There is also a software delay from the time an up-time begins and the first batch of packets being handed off to the NIC. To solve these issues, we provide a *guard time* and an *up-time preallocation* value to a TDMA module.

The guard time value is used to "cut off" an up-time early to ensure that no packets get transmitted during the down-time. We evaluate the measured loss of various guard times in Section 4.3.1. The up-time preallocation value starts an up-time early in order to ensure packets fill as much of the up-time as possible. The combined effect of these two values is visualized in Figure 3. We show two up-times along with the down-time between them.

By using multiple cores each with a dedicated TDMA module, we can create multiple "virtual hosts" that are all controlled by a single sNIC. Each virtual host can then act in isolation, generating its own flows and sending to their own distinct destinations. For our results, we create pairs of virtual hosts that only communicate with one another

Figure 3: A TDMA schedule. Up-time preallocation and guard time correct for observed delays and variance in packet transmission from NIC hardware.

on fixed up/down intervals in order to provide a baseline evaluation of the capabilities of sNICs for TDMA scheduling.

Because BESS will run each module at full speed unless otherwise specified, we use a rate limiting scheduler with the BESS module as its child to ensure virtual hosts do not compete with one another for link bandwidth.

3.4 Multi-Hop Indirection

We measure the latency and accuracy of multi-hop indirection in sNICs to understand if they can be utilized for future datacenter architectures. We use packet encapsulation, which provides the most flexibility but slightly larger overhead due to the additional data and processing required for each packet in a flow. Another method of indirection could be to communicate a forwarding pattern as a scheduling state to the TDMA module, but this has extremely limited flexibility within our specific implementation.

BESS supplies generic header encapsulation and decapsulation modules to the user, and it can be used to implement both standard GRE headers as well as any arbitrary header the user wishes to define. DPDK packets provide a buffer on either side of the packet when they are allocated in order to rapidly prepend and append information without requiring the entire data buffer to be copied.

We created a simple forwarding application using basic modules that encapsulates each packet of a single flow with a 4-byte standard GRE header and timestamps packets before sending them to a receiving host. The receiver then removes the GRE header, swaps the MAC addresses in the Ethernet header, and then forwards the packet back to the sending host. The sending host records round-trip packet latency.

3.5 Statistics Modules

We created a few modules to gather statistics for our experiments. While basic rate information is provided by BESS, we wish to inspect interpacket gaps for packet pacing and record arrival times to analyze the performance of our TDMA module and multi-hop indirection application. We leverage the histogram object provided by BESS for TDMA scheduling statistics and round-trip latency.

Recording interpacket gaps cannot be done precisely in software, and we do not have a hardware solution available for accurately timestamping packets at 100-Gb/s. Instead, we created a module that records the arrival time and size of each batch of packets for a flow in a memory array for a fixed number of batches. This is then dumped to a file at the end of an experiment for later analysis.

While loss information can be provided by NIC counters, we created a sequencing module that detects packet loss in a flow. The sequencing module has a paired loss detection module that simply inspects the sequence number of each packet and records any gaps in a histogram. We slightly modify BESS to allow the sequencing module to retransmit sequence numbers if they were dropped at the sender by a later module.

4 RESULTS

We now present results when using sNICs for a variety of flow control components along with microbenchmarks to provide an understanding of software NIC baseline performance. We execute our tests on a pair of Dell PowerEdge R630 servers, each with a 100-Gb/s ConnectX-5 NIC. The NICs are connected together using a 100-Gb/s Mellanox Spectrum Ethernet switch. Both servers are configured with two 12-core Intel Xeon E5-2650v4 CPUs, though only one is used

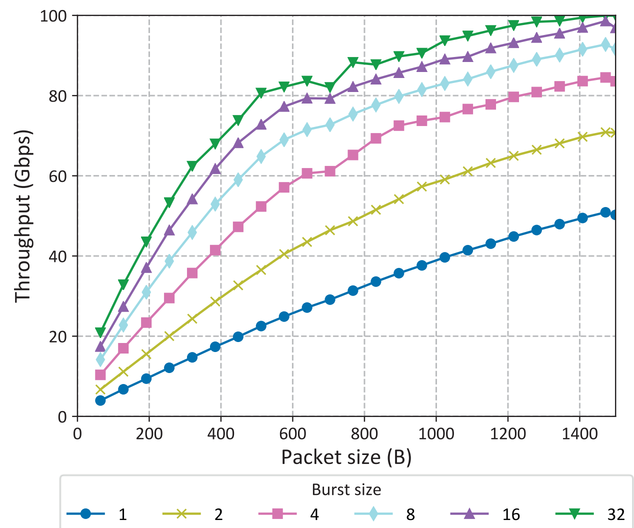


Figure 4: sNIC data throughput with one sending core.

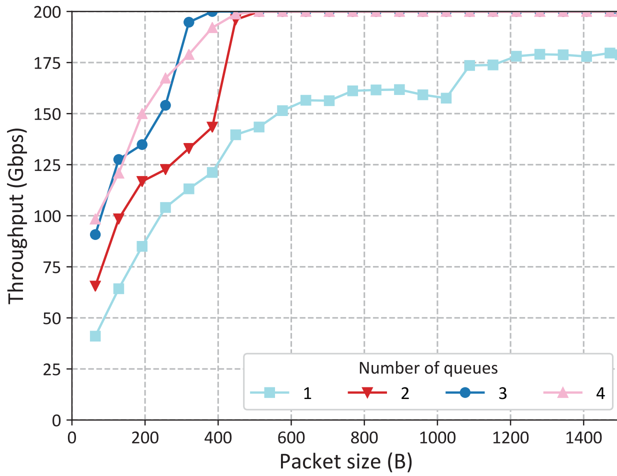


Figure 5: Combined sending/receiving throughput using bidirectional flows with a burst size of 32 packets.

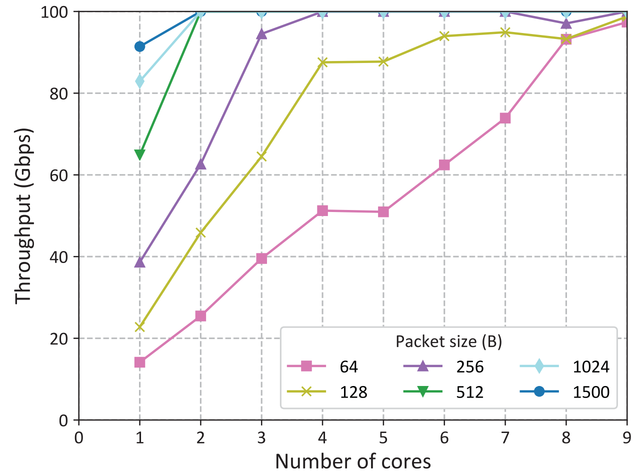


Figure 7: sNIC throughput with an 8 packet burst size.

to ensure we do not require any tasks to use QPI to communicate with the NUMA domain of the NIC. Our servers run Ubuntu 16.04.3 LTS with Linux kernel version 4.4. We use a current open-source version of BESS that leverages DPDK version 17.11. The ConnectX-5 network card driver and firmware is provided by MLNX_OFED version 4.2-1.2.

4.1 Microbenchmarks

We begin by executing a few microbenchmarks that allow a basic understanding of the baseline performance of software NICs. This is a requirement when properly evaluating more complex applications, as we must understand if failures and limitations observed there are a result of one or more basic

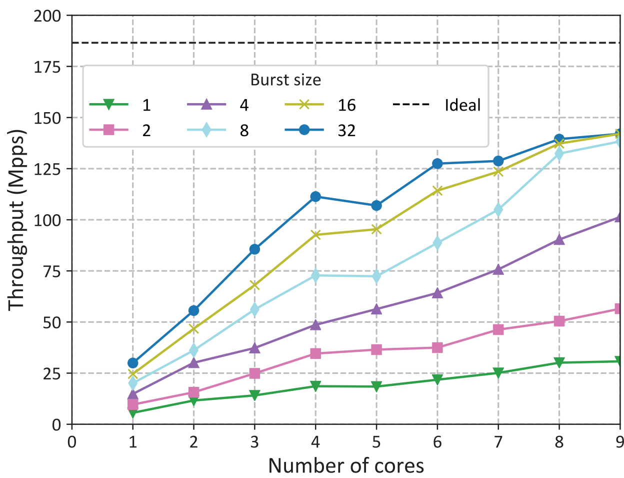


Figure 6: sNIC throughput with 64-byte packets.

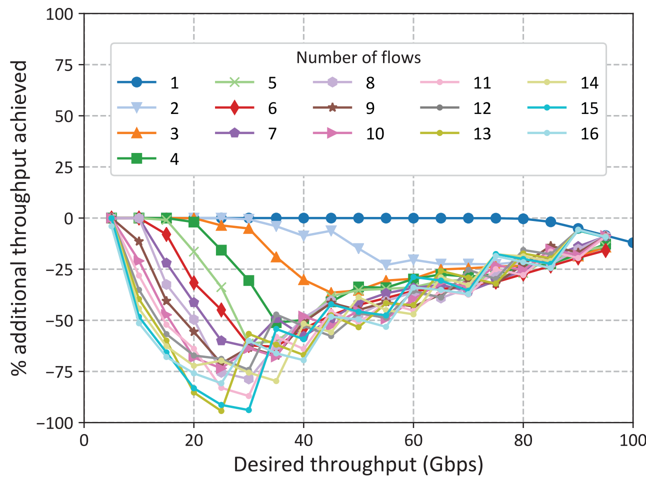
restrictions. In all but one microbenchmark, one host acts as a dedicated sender, and the other as a dedicated receiver. We report the statistics at the sender, as the receiver was not the limiting factor in our microbenchmarks.

We first determine the throughput that a single core can provide in gigabits per second (Gbps) as a function of packet size. We also vary the burst size, which determines how many packets are created and sent in a single scheduling loop. A lower burst size means more time will be spent between allocating and sending each packet, but allows for finer grained rate limiting and packet pacing.

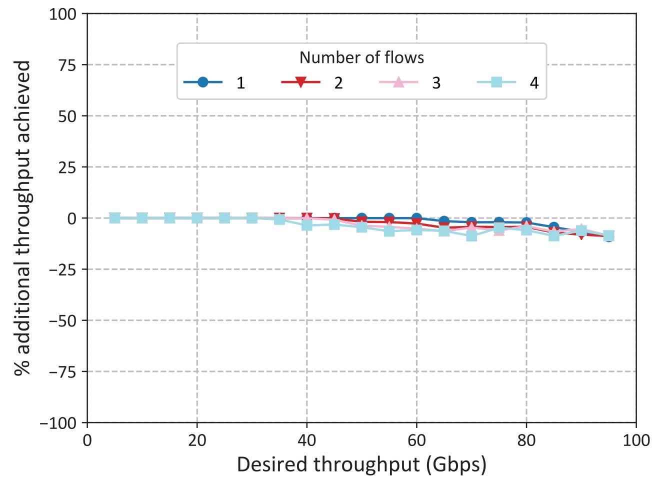
The results in Figure 4 suggests that a single core is able to saturate the entire link if maximum sized packets and the largest possible burst size (32 packets) is used. Even still, saturating the link with 40Gbps of bandwidth is very feasible with a large range of packet and burst sizes. As a baseline, a single core is quite capable of saturating link bandwidth. At 100Gbps, saturating with a single core is typically infeasible, but given 100Gbps is a large amount of data for a single core to handle in many scenarios, this is neither unreasonable nor unexpected.

Some of our later experiments use bidirectional flows, where each host acts as both a sender and receiver. We modify our previous microbenchmark and have each machine have a number of sending and receiving queues simultaneously handled by different CPU Cores. We measure the combined sending and receiving throughput at a single host.

We see from the results in Figure 5 that when a single queue is simultaneously sending and receiving data, it is unable to achieve 100-Gb/s bidirectionally. We are unable to determine whether this is a limitation of BESS or DPDK, but this does mean that users should be aware that multiple



(a) Flows from a single core.



(b) Flows from four cores.

Figure 8: Rate limiting accuracy of a primary flow. Remaining bandwidth is evenly split across all other flows (not shown). All flows use 1500 byte packets and a burst size of 8 packets.

queues may be required to achieve 100-Gb/s bidirectionally on an endhost.

Our next microbenchmark examines sending throughput and how this scales with multiple sending queues, where each queue is dedicated to a single physical CPU core. We measure the sending rate at the receiver in millions of packets per second (MPPS). We use 64 byte packets in order to ensure that the link can be saturated with the maximum possible number of packets instead of data. We again test with different burst sizes at the sender.

The results are presented in Figure 6. We note that the advertised maximum MPPS for the NIC we use is roughly 140 MPPS [19], meaning the theoretical ideal maximum shown in the figure is likely not achievable with our hardware. With the maximum possible burst size (32 packets), we can achieve just over 142 MPPS.

For our TDMA experiments in Section 4.3, we select a fixed burst size of 8 packets. The final microbenchmark we perform is the throughput scaling of multiple cores with this same fixed burst size and different packet sizes. We again have each queue on unique physical CPU cores.

The data in Figure 7 asserts that demonstrate that adding a small number of additional cores can linearly scale throughput for all packet sizes, but as more cores are added, throughput grows at a slightly slower rate. Interestingly, throughput always increases when using 64 byte packets, but otherwise there is a slight loss of performance when using 8 cores. Our hypothesis is again that there is some form of CPU cache contention at this point that has a more negative effect than the increased throughput provided by an additional core.

We found it interesting that performance decreased when increasing the number of sending queues or packet size in a few places, and are not able to provide a definitive explanation as to why. Our hypothesis is that at these points the CPU cache became limited due to the increased memory required, offsetting the potential benefit.

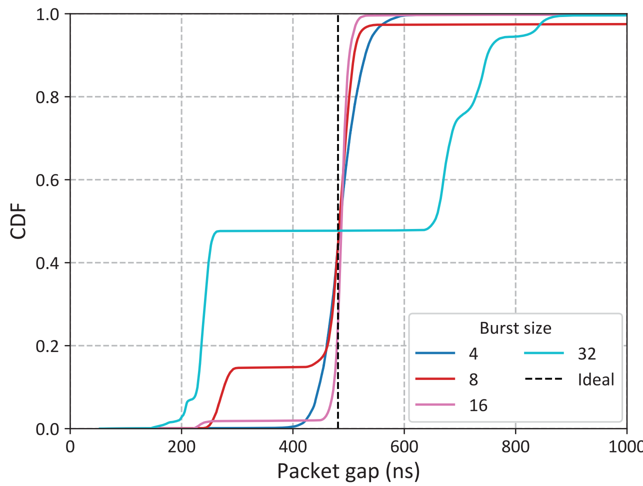
We see from these microbenchmarks that although we are unable to completely saturate a 100-Gb/s link with non-MTU size packets, it can send at a significant fraction of that with only a few cores and a moderate burst size. However, in order to ensure link saturation for future experiments with a lower number of queues, we use packet sizes of 1500 bytes.

4.2 Rate Limiting

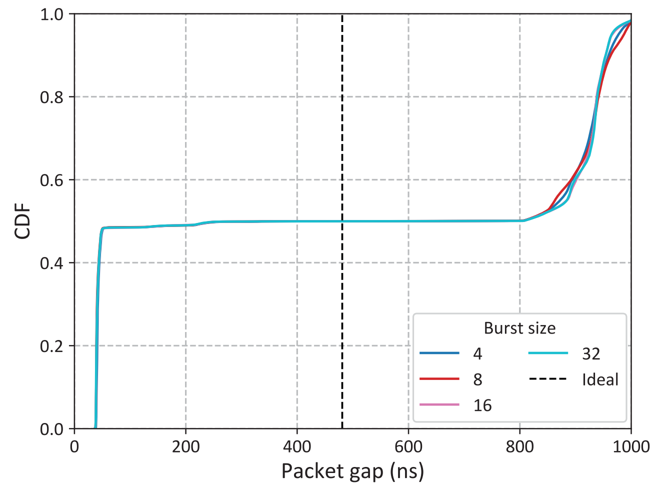
We examine the accuracy of sNICs for rate limiting over a number of different active queues and flows per sending queue, using one dedicated core per queue. We again have dedicated sender and receiver hosts. We use rate increments of 5Gbps and record the amount of bandwidth observed at the receiver. We create one receiver queue per sending flow to ensure the receiver accurately gathers sending statistics.

We sweep the desired rate for the first (primary) flow on the first queue we create. The remaining rate on the link is distributed evenly across every other flow in the experiment, including flows sent from other queues. We record both pacing and loss data along with the average throughput of each flow (see below).

We first increase the number of flows that are sent from a single core (and thus queue) and compute how close the achieved rate is to the desired rate for that flow. We use a



(a) A single flow.



(b) Four queues, each with one flow.

Figure 9: CDF of estimated inter-packet gaps with 1500 byte packets for a flow. All flows are limited to 25Gbps.

burst size of 8 and a packet size of 1500 bytes in order to provide a balance between good throughput of single and multiple flows.

The clear takeaway from the data in Figure 8a experiments is that multiple flow tasks with different rate limits do not perform well on a single queue. With just two flow tasks sharing the link bandwidth the primary flow underperforms, even though with 1500 byte packets it is possible for a single flow to almost completely saturate the 100-Gb/s link. However, we see that the primary flow never takes more bandwidth than it is allotted, which is a desirable invariant.

In order to further understand this issue, we perform the same test with four active queues and between one to four flows per queue. This means there are again at most 16 flows sending at any given time. Bandwidth not assigned to the primary flow is again split evenly across every flow across all queues.

The performance seen in Figure 8b is much more promising. We immediately see that while we are still unable to completely fulfill requested rate limits for high amounts of bandwidth, there is not a rapid collapse between the achieved versus requested rate. This argues that the sNIC is currently unable to handle multiple rate limits on a single core when that core is responsible for sending extremely large amounts of traffic, and does far better when some of the link bandwidth is offloaded onto other cores.

4.2.1 Packet Pacing. During our rate limiting evaluations, we record the arrival time and size of each batch of packets for each flow at the receiver. We then assume that each packet in a batch was evenly spaced between the arrival time of that batch and the previous batch. While this is an optimistic evaluation of packet arrival times for a flow, it does permit us to coarsely inspect the distribution of inter-packet gaps for each flow without requiring an advanced hardware solution. We ensure that each receiver core only handles one flow.

We again present results for a "primary" flow that is allocated a fraction of the overall link bandwidth on one core, with the remaining bandwidth distributed across all other flows both on the same and other cores. We plot the arrival times as a cumulative distribution function, with the ideal interpacket gap for the given bandwidth displayed as a vertical dotted line.

Our results when sending with only a one flow and one queue at 25Gbps are plotted in Figure 9a. This exhibits that up to moderate burst size of 16 packets, the sNIC is capable of pacing packets to a reasonable degree of error, some of which may be attributed to our measurement method.

We've established previously that we need to send with multiple cores and hardware queues in order to permit multiple flows to fulfill requested rate limits. However, when increasing to sending with four separate queues with each sending at 25Gbps, we are now reliant on the NIC hardware to properly stripe sending across each hardware queue, rather than batching groups of packets from each queue together. As we see in Figure 9b, our NIC clearly does the latter and the burst size we configure no longer affects the distribution of interpacket gaps. This behavior may play a role as to why multiple queues are better able to fulfill requested rate limits.

4.3 Basic TDMA Schedules

Our investigation into the performance of TDMA schedules on sNICs is done by sweeping a few different basic parameters in order to gain insight into the core limitations of software NICs. The results here are targeted towards the TDMA specific requirements described in Section 3. Because more complex TDMA schedules will have strict requirements in order to work effectively, we work here to identify scenarios where the performance of a software NIC is insufficient to identify future work for both networking software and hardware to satisfy future systems.

We run each test with a number of virtual TDMA hosts on the same two physical machines, with each machine hosting half of the virtual hosts for the experiment. Each virtual host requires two physical CPU cores; one for sending scheduled traffic, and the other for receiving traffic and gathering statistics. We use a third physical core for each virtual host to generate traffic to ensure infinite demand. Each virtual TDMA host only sends a single flow to one other virtual host in a repeating static up/down pattern, as described in Section 3.3.1. We use a fixed up-time preallocation value of 3 microseconds, which we determined based on preliminary results. The burst size of each virtual host is fixed at 8 packets to balance throughput and emission accuracy. We use dummy packets sized at 128 bytes, which are able to mitigate NIC transmission delays while not creating any observable overhead on performance.

4.3.1 Guard Time. As discussed in Section 3.3.1, the appropriate guard time value varies due to both software and hardware. Because evaluating the effect of hardware in our case would require multiple 100Gbps NICs to cross reference their minimum appropriate guard time value, we instead simply present a sweep over our experimental setup to evaluate the effect of different guard times on the quantity of packets seen during down-time. We use 8 virtual hosts (4 per physical machine) rate limited to 10Gbps each.

Our results are shown on Figure 10. While it may initially seem unintuitive that larger packet sizes are lost more often, this is caused by batches of packets transmitting just before the up-time ends, meaning some fraction of the batch will be lost. If the batch takes more time to transmit, more of the overall batch will end up being transmitted during down-time. There is observable variance in the results caused by the hardware and scheduling randomness. We see that a guard time of 4 microseconds is sufficient for preventing packets of any size from arriving during down-times. For the following results we selected a value of 5 microseconds in order to provide a reasonable buffer and eliminate any possible variance in the transmission delay of packets that may cause packets to be transmitted during down-times.

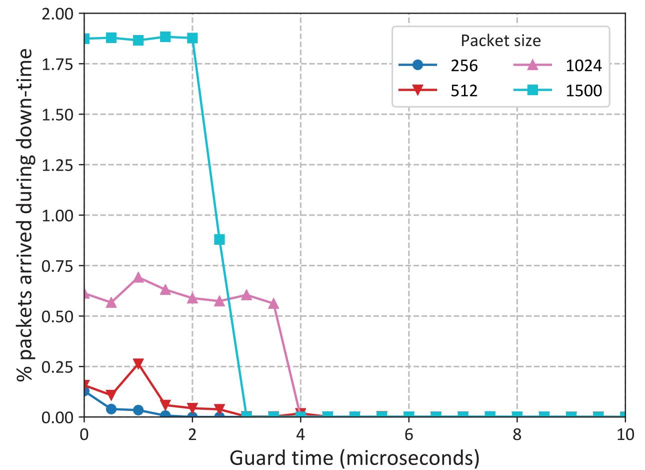


Figure 10: Effect of guard time on the amount of packets observed during down-times with 8 virtual hosts each sending at 10Gbps.

4.3.2 Duty Cycle. The purpose of a TDMA schedule's duty cycle is important to determine how to appropriately balance the schedule of flows, so it is important that a software NIC be able to support a range of up/down-time durations without sacrificing accuracy or performance. This becomes especially important if dynamic schedules are used, as the duration of the up-time or down-time will vary between shorter and longer values over time.

We fix the duration of the TDMA scheduling period and use a number of different up/down-time length ratios to determine the effect that the duty cycle may have on both achieved rate and packet loss. We compute the packet loss rate across all virtual hosts via both dropped packets via sequence numbers and observed down-time packets. We use a period of 500μs and limit each virtual host to 25 Gbps.

We see from Figure 11 that with both small up-times and down-times packet loss begins to increase (thus throughput suffers), and shorter down-times have a greater effect than shorter up-times. When the number of virtual hosts combined leverages all the available link bandwidth, a large amount of packet loss occurs and throughput suffers. With shorter down-times, we note that a reasonable fraction of the loss is caused by packets arriving during down-times. Interestingly, having a small number of virtual hosts that leverage a smaller fraction of the link bandwidth avoids most of the performance issues observed with higher speeds and more hosts, although shorter up-times do seem to induce a small penalty.

4.3.3 Scheduling Period. We next examine the performance of TDMA when the duty cycle is fixed, but the scheduling period is varied. We briefly mentioned in Section 3.3.1

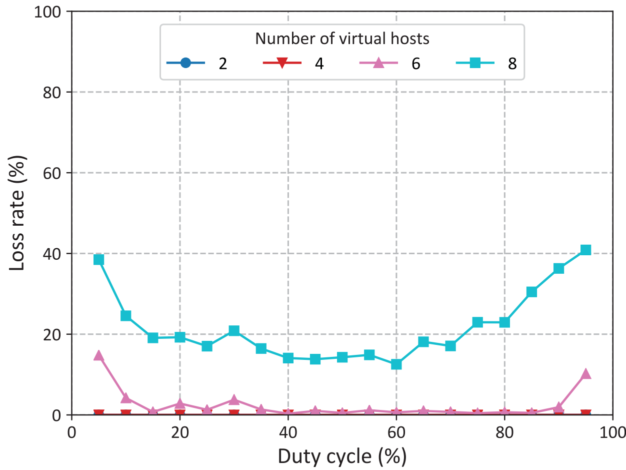


Figure 11: TDMA packet loss rate by duty cycle with a period of 500μs and 25 Gbps per virtual host.

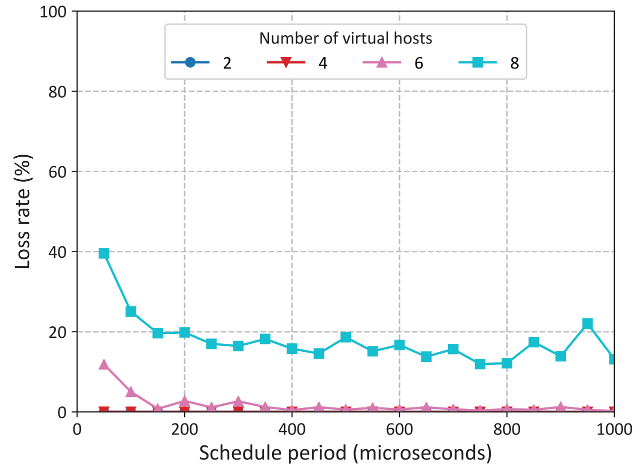


Figure 12: TDMA packet loss rate by scheduling period with a duty cycle of 50% and 25 Gbps per virtual host.

that smaller scheduling periods allow new flows to be scheduled sooner, as they have to wait less time for the previous period to complete before they can begin sending traffic. Additionally, longer periods can be desirable in order to avoid down-time weighted duty cycles caused by high reconfiguration delays of physical networking hardware, such as 3D-MEMS optical switches.

We again fix the sending rate of each virtual host to 25Gbps, and use a fixed 50% duty cycle with a varying scheduling period. We compute statistics as in the duty cycle experiments.

In Figure 12, we observe similar trends with the number of active virtual hosts from the duty cycle experiment with no significant deviations. We notice that small scheduling periods still seem to cause significant packet loss at the receiver. The variance in performance becomes larger as the period increases at higher bitrates due to the increased duration that the sNIC must saturate the link, which we discovered can be difficult in Section 4.2.

4.4 Multi-Hop Indirection

To test multi-hop indirection, we create a single sending queue with one flow sending 1500 byte packets at various rate limits to understand how forwarding latency is affected by network speed.

While rate limiting performance for a single queue and flow performed reasonably well in our rate limiting experiments, it is expected that any performance degradation caused by rate limiting will affect forwarding latency in any system. Because multi-hop indirection requires sacrificing the forwarding host's outgoing throughput on indirected traffic, it is typically necessary to rate limit forwarded data so that it does not completely consume the outgoing link's available bandwidth.

The results of our experiment are in Figure 13. The boxes represent the standard quartiles for the results, and the whiskers represent the 0.1 and 99.9 percentiles of observed latency to account for scheduling variance during beginning the test and while collecting results. We recorded loss information (not shown), and measure roughly 18% packet loss only at speeds of 80Gbps and above, explaining the massive spike in latency at those points.

We see that the forwarding latency does slightly increase for higher speeds before loss occurs, implying that there is some software-associated overhead with forwarding encapsulated packets at high network speeds. We also observe that the forwarding latency at low speeds is actually rather large. We suspect this is due to the hardware NIC delaying packet transmissions until a sufficiently large batch has been enqueued, which explains the large gap between the median and the minimum values.

4.5 sNIC Performance Observations

Although these tests are not completely exhaustive, we have gathered enough results that we have gained a basic understanding of the limitations of sNICs for usage in a few different flow control contexts. We do not expect these results to be a statement on whether the idea of software NICs is well formed or invalid, but just an observation of the current state of affairs based on the results we have presented.

The amount of bandwidth that an sNIC can provide to the network is very good for 40-Gb/s links: only a few cores are necessary to send at 40-Gb/s with minimum size packets. However, scaling to 100-Gb/s speeds becomes difficult for

smaller packet sizes without using a large number of cores, which is prohibitive for non-experimental usage. We suspect that the microbenchmarks perform roughly equivalent to base DPDK due to the simplicity of the sNIC application used to run them, implying that the core technologies enabling software NICs at all may need further development in order to satisfy future networking demands.

Per-flow rate limiting performance in the sNIC was rather disappointing when using a single core. We note that given the performance of the microbenchmarks that this is primarily due to the large amount of bandwidth sent on a single CPU, especially since splitting the bandwidth across multiple CPUs drastically improved performance.

While packet pacing performance was disappointing, it was not unexpected. It is impossible for current software to control the exact method that the NIC firmware will use to select and send packets from various queues. We are impressed that the sNIC and DPDK managed to pace a single flow fairly well in isolation, given that the hardware NIC could have simply held onto packets until a minimum size threshold. Future hardware could provide a method through DPDK to configure some of the firmware's packet transmission parameters to make software-programmed packet pacing a viable alternative. At the moment, hardware is still a necessary solution to enforce pacing of flows.

The TDMA performance of the sNIC is satisfactory when it is bound within the limits found in the previous results and the parameters of the schedule are within some moderate restrictions. The only serious setback for TDMA is that scheduling periods at or less than 50μs begin to incur serious performance setbacks at higher speeds, which makes multi-hop indirection more essential when using sNICs for TDMA flow control.

Multi-hop indirection proved to have higher latency than we expected in even the best case, though we are unable to determine how much of this is due to software versus hardware. The current performance is sufficient to reduce flow latencies for TDMA based on the required scheduling periods we observed. However, many datacenter networks are beginning to use low flow latency as a primary metric, and the multi-hop forwarding latency of the sNIC is an order of magnitude higher than current expectations for flow latencies in datacenters [32].

Additionally, the limited flexibility of the scheduling subsystem discussed in Section 3.3.1 prevents us from seeing sNICs as a perfect solution for future-proof TDMA flow control development. Nonetheless, the module subsystem is easily extensible and provided sufficient performance with a limited number of cores at 40-Gb/s that we can highly recommend that users leverage sNICs in current datacenter networks in endhosts for a range of flow control systems.

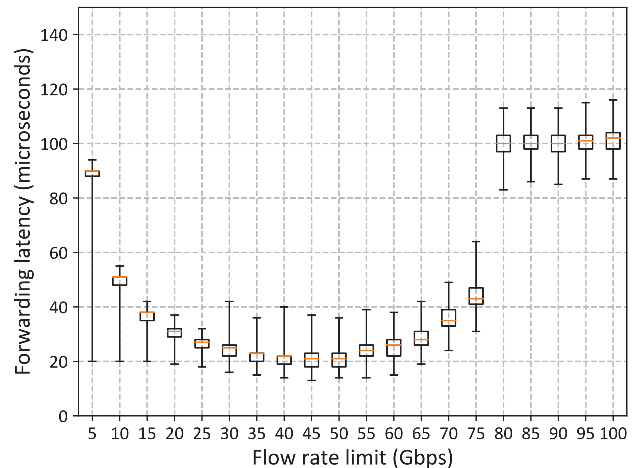


Figure 13: Packet forwarding latency using GRE encapsulation with 1500 byte packets and a single flow.

5 CONCLUSION

Software NICs provide a flexible framework to aid to future flow control development, and their features aim to supplant some of the trends towards offloading networking functions onto NIC hardware. Rate limiting, packet pacing, and TDMA flow scheduling, and multi-hop indirection are traditionally difficult flow control problems for software, despite how prevalent they are in many of today's flow control systems.

BESS, a software NIC developed specifically for endhosts and built on the DPDK software library, provides a modular system that we use to implement and measure the performance of these forms of flow control at 40- and 100-Gb/s speeds. While sNICs allow for rapidly evolving applications with satisfactory performance for 40-Gb/s networks, there is still work to be done in order for sufficient performance in 100-Gb/s environments.

6 ACKNOWLEDGMENTS

This work was funded in part by the National Science Foundation (CNS-1564185, CNS-1629973, and CNS-1553490). Thanks to Alex Forencich, Alex Snoeren, and William M. Mellette for providing important feedback throughout this project. Additionally, we would like to extend special thanks to all of the BESS developers, particularly Barath Raghavan and Sangjin Han, for providing both support and invaluable guidance on using the BESS software architecture.

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[2] Mohammad Alizadeh, Shuang Yang, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Deconstructing datacenter packet transport. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 133–138, New York, NY, USA, 2012. ACM.

[3] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. SIGMETRICS '16.

[4] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, and Q. Dong. WaveCube: A scalable, fault-tolerant, high-performance optical data center architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1903–1911, April 2015.

[5] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, and Xitao Wen. OSA: An optical switching architecture for data center networks and unprecedented flexibility. NSDI '12.

[6] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 407–424, Santa Clara, CA, 2016. USENIX Association.

[7] Yong Cui, Shihan Xiao, Xin Wang, Zhenjie Yang, Chao Zhu, Xiangyang Li, Liu Yang, and Ning Ge. Diamond: Nesting the data center network with wireless rings in 3D space. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 657–669, Santa Clara, CA, 2016. USENIX Association.

[8] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. SIGCOMM '10.

[9] The Linux Foundation. Data Plane Development Kit. https://dpdk.org/.

[10] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. SoftNIC: A software NIC to augment hardware. Technical Report UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.

[11] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 29–42, New York, NY, USA, 2017. ACM.

[12] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, and Changhoon Kim. EyeQ: Practical network performance isolation for the multi-tenant cloud. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[13] Xin Jin, Nathan Farrington, and Jennifer Rexford. Your data center switch is trying too hard. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 12:1–12:6, New York, NY, USA, 2016. ACM.

[14] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 281–294, New York, NY, USA, 2017. ACM.

[15] Antoine Kaufmann, Simon Peter, Thomas Anderson, and Arvind Krishnamurthy. FlexNIC: Rethinking network DMA. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

[16] R. Kawashima, H. Nakayama, T. Hayashi, and H. Matsuo. Evaluation of forwarding efficiency in NFV-nodes toward predictable service chain performance. *IEEE Transactions on Network and Service Management*, 14(4):920–933, Dec 2017.

[17] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit Switching Under the Radar with REACToR. NSDI '14.

[18] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. Scheduling techniques for hybrid circuit/packet networks. CoNEXT '15.

[19] Ltd. Mellanox Technologies. Mellanox sets new DPDK performance record with ConnectX-5. http://ir.mellanox.com/releasedetail.cfm?ReleaseID=1014514.

[20] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. RotorNet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 267–280, New York, NY, USA, 2017. ACM.

[21] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: RTT-based congestion control for the datacenter. *SIGCOMM Comput. Commun. Rev.*, 45(4):537–550, August 2015.

[22] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. NetBricks: Taking the V out of NFV. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 203–216, Berkeley, CA, USA, 2016. USENIX Association.

[23] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 307–318, New York, NY, USA, 2014. ACM.

[24] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. SENIC: Scalable NIC for end-host rate limiting. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 475–488, Berkeley, CA, USA, 2014. USENIX Association.

[25] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 123–137, New York, NY, USA, 2015. ACM.

[26] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 404–417, New York, NY, USA, 2017. ACM.

[27] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 309–322, Berkeley, CA, USA, 2011. USENIX Association.

[28] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs HÃűlzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. In *Sigcomm '15*, 2015.

[29] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 205–219, New York, NY, USA, 2016. ACM.

[30] Bhanu Chandra Vattikonda, George Porter, Amin Vahdat, and Alex C. Snoeren. Practical TDMA for datacenter ethernet. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 225–238, New York, NY, USA, 2012. ACM.

[31] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: Part-time optics in data centers. SIGCOMM '10.

[32] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 139–150, New York, NY, USA, 2012. ACM.

[33] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao, and Haitao Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. SIGCOMM '12.