MIMIC: SmartNIC-aided Flow Backpressure for CPU Overloading Protection in Multi-Tenant Clouds

Enge Song*, Nianbing Yu*, Tian Pan*, Qiang Fu†, Liang Xu*, Xionglie Wei*,
Yisong Qiao*, Jianyuan Lu*, Yijian Dong*, Mingxu Xie*, Jun He*, Jinkui Mao*,
Zhengjie Luo*, Chenhao Jia*, Jiao Zhang‡, Tao Huang‡, Biao Lyu§*, Shunmin Zhu¶*
*Alibaba Group †RMIT University ‡Purple Mountain Laboratories §Zhejiang University ¶Tsinghua University

***Abstract*—In multi-tenant clouds, off-the-shelf x86 boxes are widely deployed as middleboxes. With the rapid growth of cloud traffic and the migration to NFV deployment in recent years, CPU overloading at middleboxes becomes more of an issue. From our data centers, we observed that the CPU overloading was caused by heavy hitters. To address this issue, we propose MIMIC, a cloud-scale flow backpressure system, implemented onto our existing SmartNIC with FPGA acceleration. MIMIC rate-limits the selected heavy hitters through a new per-flow backpressure protocol and a new heavy-hitter detection system, to protect the other tenants. The detection system is based on hierarchical memory design, leveraging on-chip SRAM and off-chip DRAM, which can handle highly concurrent cloud traffic without the losses of flow information. We extend the design by adding a pre-filtering procedure for rapid detection. To avoid CPU being flooded by FPGA through frequent heavy-hitter reporting, due to their performance disparity, the CPU queries the FPGA on demand. The backpressure protocol is non-invasive to protect tenant privacy and allows controllable rate-limiting through the novel use of ECN and meter tables. The SmartNIC acts as a man in the middle to facilitate heavy-hitter detection and per-flow backpressuring. In a production setting, we observe that MIMIC can react quickly and bring down CPU load to the normal level within 10ms without packet losses.**

I. Introduction

Cloud services have been increasingly adopted by organisations and individuals [1]. This trend is accelerated by the roll-out of 5G mobile networks and the pandemic. Cloud vendors are motivated to provide a diverse range of services appealing to customers. As a major cloud vendor, we have observed the rapid growth of cloud traffic in recent years. We start experiencing more often CPU overloading at our NFV-based middleboxes on the path of east-west traffic (VM-VM) (Fig. 3). By looking into individual CPU overloading occurrences, we found that CPU overloading was caused by heavy-hitter flows (Top-1 flows in Fig. 2). In a multi-tenant cloud, due to infrastructure sharing, CPU overloading may lead to packet losses and long latency across all the tenants whose traffic goes through the CPU, even if the majority of the tenants do not contribute to the heavy hitters.

CPU overloading at middleboxes is now a particular issue as we are migrating from bare-metal x86 to NFV (Network Function Virtualization) based deployment where VNFs (Virtual Network Functions) are built above NFV hypervisors such as KVM [2] (Fig. 3). Network functions such as load balancing, private gateway, firewall, VPN and NAT (Network Address Translation) are getting deployed in the NFV nodes for its programmibility, flexibility and rapid deployment [3], [4]. However, all these conveniences are achieved at the cost of performance – CPU overloading becomes more of an issue. This is in line with our experience of actual deployment (Fig. 1). Our middleboxes follow the run-to-completion model [5], hashing the flows randomly onto different CPU cores via RSS (Receiver-Side Scaling) for load balancing [6]. The RSS-based flow hashing ensures the affinity between the flow and the core, and thus fast and in-order packet processing. However, this affinity may lead to CPU core overloading in the presence of heavy hitters.

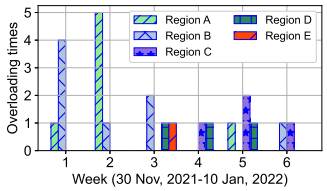


Fig. 1. CPU overloading occurrences per week across different regions.

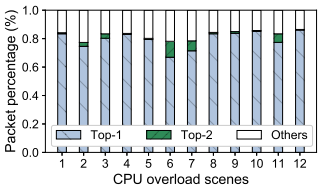


Fig. 2. CPU overloading by heavy hitters (noticeably Top-1 flows).

There are three possible ways to tackle this problem: scaling up middlebox capacity, load balancing and backpressuring. Our experience shows that while a CPU core is being overloaded by a heavy hitter, the other cores still have capacity not utilized. Scaling up middlebox capacity is not an efficient solution. On the other hand, the SmartNIC with FPGA acceleration has the same problems as Tofino: limited on-chip resources [7]. Besides, VNFs are highly customized and diverse according to the users' requirements, which makes it difficult to configure hardware logic in advance. Load balancing is another way to handle this issue. Stateful flow mapping may map a potential heavy hitter to the CPU core with the lightest load, while flow migration may reallocate a heavy hitter from one core to another [8]. However, the RSS-based flow hashing works reasonably well in terms of load balancing among the cores in the absence of heavy hitters, due to the effect of multiplexing. In either case, the "unlucky" core that gets the heavy hitter may get overloaded. Naturally we may break up the heavy-

This work was supported by Alibaba Group through Alibaba Innovative Research Program. Co-corresponding authors: Tian Pan, Xionglie Wei and Shunmin Zhu.

978-1-6654-8234-9/22/$31.00 ©2022 IEEE

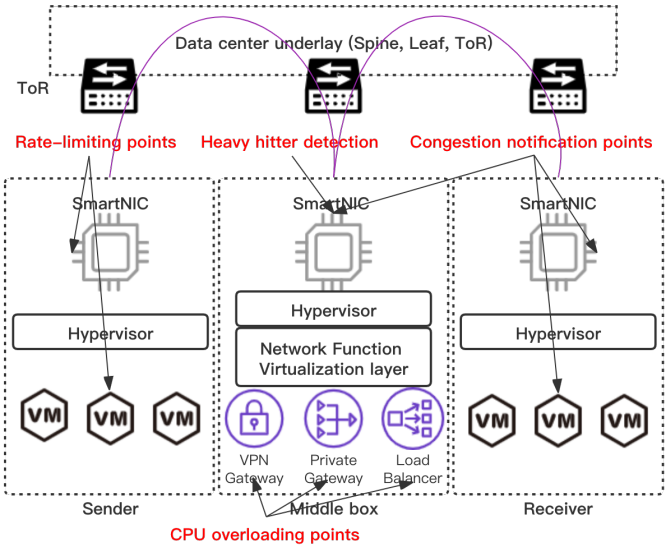


Fig. 3. Design overview. CPU overloading may happen at middleboxes.

hitter flow into pieces (flowlets or packets), and then distribute the pieces among the cores. While it may sound plausible, there are practical issues. Packets would likely be processed and then arrive at TCP receiver out-of-order, causing TCP retransmissions. Reordering has to be done before sending out the packets. Furthermore, the VNFs in the middlebox often have to maintain the state of individual flows. In this case, state sharing among the cores has to be maintained, imposing another challenge in addition to reordering. All these are not trivial tasks in a high-performance setting.

A sensible solution is backpressuring: identify the troublesome heavy hitters and then rate-limit them to protect the other tenants in the network. To this end, we propose a cloud-scale flow backpressure system, called MIMIC, which is developed onto our existing production SmartNIC with FPGA acceleration. The basic idea is to (1) trigger heavy hitter detection at the middlebox when the CPU utilization exceeds a threshold, and (2) backpressure the heavy-hitter flow to the traffic source. The SmartNIC acts as a man in the middle to facilitate heavy hitter detection as well as backpressuring selected heavy hitters. To minimize the impact on the other tenants and packet losses, rate-limiting is carried out at the source node. Based on this, we designed a **per-flow backpressure protocol**, which is capable of:

- **Selecting the appropriate flows for rate-limiting.** Among the reported heavy hitters, only the ones contributing significantly to the overloading are selected.
- **Controllable rate-limiting.** The heavy hitters are rate-limited to a desired level through the manipulation of meter tables as well as ECN marking frequency for TCP congestion control.
- **Being transparent to the hosts by "deceiving" ECN.** The ECN capability is enabled at SmartNIC without modifying the host as long as the host is willing to enable ECN when requested (default in Linux 3.x [9]).

The existing SmartNIC has the ability to keep flow statistics for all ongoing flows for monitoring, auditing and accounting services. This is achieved through a *hierarchical* memory system by taking advantage of the on-chip SRAM and the off-chip DRAM. On top of that, we designed a **heavy hitter detection mechanism** by adding a light-weight heavy hitter *pre-filtering* procedure. Through the pre-filtering, a set of ongoing flows are selected as potential heavy hitters, ready to be used by the backpressure protocol, which then determines the heavy hitters to rate-limit. To avoid CPU being flooded by FPGA through frequent heavy-hitter reporting due to their performance disparity, the CPU queries the FPGA on demand.

As a result, MIMIC can achieve a fast reaction time to rate-limit heavy hitters to the desired level across a wide range of scenarios. Our main contributions are summarized as follows:

- We propose a new heavy-hitter detection system based on hierarchical memory design, which results in no losses of flow information for improved detection accuracy. With the support of the pre-filtering procedure, the heavy hitters can be determined within 1ms.
- We propose a per-flow backpressure protocol, which can pinpoint the troublesome heavy hitters. It is non-invasive, as no modification is needed to the host. "Deceiving" tricks are done at SmartNIC to enable ECN, but only applied if the host is not against the use of ECN. It allows controllable rate-limiting through the manipulation of meter tables and ECN marking frequency for both TCP and UDP traffic.
- We extend our production SmartNIC, facilitating accurate heavy-hitter detection and swift per-flow backpressuring. The additional pre-filtering procedure only requires two Pingpong tables in SRAM to keep the FlowID of the potential heavy hitters.

MIMIC has been deployed in our data centers to resolve CPU overloading at middleboxes caused by heavy hitters. It can bring down CPU load to the normal level within 10ms with no or almost no packet losses.

II. Design Overview

MIMIC is designed with two modules: 1) the end-to-end per-flow backpressure protocol (§III), and 2) the FPGA-based heavy hitter detection module (§IV).

We extend the existing SmartNIC used by the hosts and the middleboxes in our production cloud (Fig. 3). The NFV-based middleboxes on the path of east-west traffic (VM-VM) may experience CPU overloading. This makes its SmartNIC the ideal point to detect heavy hitters. The middlebox SmartNIC and the receiver SmartNIC and VM may act as the point(s) to facilitate congestion notification, while the sender SmartNIC and VM may act as the point(s) to rate-limit the selected flows, depending on the scenarios (§III).

As shown in Fig. 4, the SmartNIC consists of three parts: CPU, FPGA and off-chip DRAM. The x86 server hosts the VMs through a hypervisor where light-weight scripts are running to monitor CPU utilization. When the CPU utilization in the x86 server exceeds the pre-defined threshold, this will trigger MIMIC on the SmartNIC (light blue arrow line in Fig. 4). The CPU will query the FPGA for heavy hitter information. The FPGA has the FlowID of all current heavy

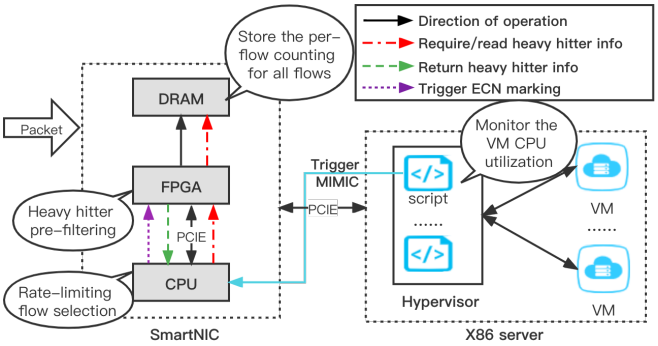


Fig. 4. SmartNIC architecture with x86 server. CPU overloading at x86 server will trigger MIMIC on SmartNIC. SmartNIC CPU gets a set of heavy hitters from pre-filtering, to be used by the backpressure protocol determining the appropriate heavy hitters for rate-limiting.

hitters in its cache obtained through pre-filtering (§IV-C). Based on the FlowID, the FPGA reads the counters of these flows stored in the DRAM. This process is shown by the red dash-dotted arrow line in Fig. 4. The FPGA then returns the heavy hitter information to the CPU (green dashed arrow line in Fig. 4) where the proper heavy-hitter flows are selected for rate limiting (§ III-A). The CPU then inserts ECN marking rules into the flow table entries for these selected heavy-hitter flows (purple dotted arrow line in Fig. 4). The selected flows are then rate-limited accordingly.

Our current production system already has per-flow counting statistics for all ongoing flows for monitoring, auditing and accounting services. In Fig. 4, this is achieved through a two-level memory design, leveraging the on-chip SRAM and off-chip DRAM. The heavy hitter pre-filtering procedure can readily utilize these statistics to pre-select a set of heavy hitters, to be used by the backpressure protocol, which then determines the appropriate heavy hitters for rate-limiting. Our design capitalizes on existing mechanisms such as TCP congestion control [10], ECN [11] and meter tables [12] for rate-limiting. MIMIC is designed in a way that it can adapt to any improvements on these mechanisms if they get deployed.

A. Design challenges and goals

Non-invasive per-flow backpressure. MIMIC only rate-limits the specified heavy-hitter flows [13] without affecting other flows to ensure tenant isolation [14], [15]. Modifications need to be transparent to users. The design should *not* modify the user's protocol stack and OS configurations, or emulate the behavior that is against the user's setting. For example, if the tenant chose to use CUBIC for congestion control, we would not emulate the behavior of DCTCP. If the tenant does not want to initiate or accept ECN, we will not emulate the behavior of ECN.

Controllable rate-limiting for both TCP and UDP traffic. TCP congestion control has its own rate control mechanism and tends to generate bursty traffic making transmission rate fluctuate. On the other hand, UDP has no built-in feedback mechanisms to react to congestion. It is essential to have a mechanism ensuring that the transmission rate of the heavy hitter is maintained at the expected level.

Accurate, lightweight and rapid heavy hitter detection. Accurate detection ensures that the right flows are rate-limited without affecting other legitimate traffic. This requires per-flow counting over a large time window so that micro-bursts can be filtered out. However, this will consume a large amount of memory, not doable with the limited on-chip memory. To achieve a fast backpressure reaction, the detection needs to be running all the time in the background handling cloud-scale traffic. However, the bandwidth disparity between SRAM and DRAM as well as FPGA and CPU may create bottlenecks with frequent updates. This requires a lightweight detection system, which is able to handle these complexities.

III. BACKPRESSURE PROTOCOL

Among the reported heavy hitters obtained from pre-filtering (§IV-C), the backpressure protocol selects the appropriate flows for rate-limiting (§III-A). We then design the protocol in a progressive fashion. BK-VM (§III-B) assumes the full support of ECN and can only work with TCP traffic. BK-NIC (§III-C) takes advantage of meter tables for rate-limiting and thus can work with both TCP and UDP traffic, but may generate a large number of packet losses, due to the nature of meter tables. BK-VM&NIC (§III-D) leverages both BK-VM and BK-NIC, but do not assume the full support of ECN. Instead, it enables ECN at SmartNIC by "deceiving" ECN as long as the host is willing to enable ECN upon request (default in Linux 3.x [9]).

A. Heavy hitter selection

Among the heavy hitters reported from pre-filtering, if we only rate-limit Top-1 flow, the released capacity can be quickly taken up by the other heavy hitters of the similar rates. On the other hand, it may not be necessary to rate-limit Top-2 flow if it is much smaller (Fig. 2). To handle this problem, we design a threshold-based algorithm to select the flows for rate-limiting. As engine CPU utilization relates more directly to PPS than BPS [16], packet counts are commonly used to indicate the size of the flows [13], [17]–[19]. In our design, we compare the packet count of an individual heavy hitter to the total packet count for all reported heavy hitters. This gives us an individual-to-all packet count ratio. The heavy hitters with a packet count ratio greater than a predefined threshold will be selected for rate-limiting. The threshold ratio can be selected based on individual production settings.

B. Backpressure to VM

Similar to ConQuest [13] and NFVnice [3] (when its service chain spreads across multiple hosts), a straightforward solution is to use ECN for end-to-end per-flow backpressure, which we refer to as BK-VM. BK-VM is based on the assumption that ECN can be established between the client and the server through negotiation. The negotiation is affected by host settings, *e.g.*, /proc/sys/net/ipv4/tcp_ecn in Linux 3.x [9]. Here is how ECN works to backpressure heavy hitters.

ECN modes and establishing ECN. If tcp_ecn is set to 0, ECN is disabled. If set to 1, the host will initiate ECN and

Algorithm 1: BK-VM deployment.

```
1  Function Select_flows():
2      Get elephant flow information from FPGA.
3      Calculate the individual-to-all packet count ratio and select the
         flow into FlowID_set whose ratio exceeds the threshold.
4      return FlowID_set
5  Function CE_labelling(packet, FlowID_set):
6      Get FlowID of the packet.
7      if FlowID in FlowID_set then
8          Set CE field in the IP header to 1.
9  Function Intermediate_NIC1(packet):
10     if CPU utilization > Threshold then
11         FlowID_set = Select_flows()
12     CE_labelling(packet, FlowID_set)
13     Forward the packet.
```

accept it if requested. If set to 2, the host will not initiate ECN but will accept if requested (default in Linux 3.x [9]). On the client side if tcp_ecn is set to 1, the client will request ECN on establishing the TCP connection by setting the Congestion Window Reduced (CWR) and ECN-Echo (ECE) field to 1 in the SYN segment. On the server side if tcp_ecn is set to 1 or 2, in response the server will set the ECE field to 1 in the SYN-ACK segment, and move its state machine in the kernel to the state TCP_ECN_OK, indicating ECN connectivity with the client. On receiving the SYN-ACK segment, the client will also move its state machine to the state TCP_ECN_OK [20].

ECN marking for congestion control. With ECN enabled, the sender marks its packets with the ECN Capable Transport (ECT) field set to 1. This allows intermediate routers that support ECN to mark those IP packets using the Congestion Experienced (CE) field instead of dropping them, in order to signal impending congestion. Upon receiving an packet with ECT and CE set to 1, the receiver echoes back the congestion occurrence with the ECE field set to 1 in its ACK. When the sender receives an ACK with ECE set to 1, it reduces its congestion window (CWND) and thus its transmission rate, and then sends packets with CWR set to 1 to acknowledge receiving the congestion indication. The receiver keeps sending ACKs with ECE set to 1 until it receives a packet with CWR set to 1.

If the CPU utilization of some forwarding instances exceeds the threshold, the protocol selects the heavy hitters to rate-limit (line 10-11 in Algorithm 1). It will then mark the CE (Congestion Experienced) of these flows and forward the packet (line 12-13). The sender will reduce its rate accordingly.

BK-VM is the simplest solution, leveraging the existing congestion control and ECN mechanisms. However, it assumes full ECN support and does not work with UDP traffic.

C. Backpressure to SmartNIC

Another solution is to use a meter table [21] for rate-limiting at the sender host. This only requires the SmartNICs at the sender and the middlebox to work together, and thus is transparent to the VMs. The middlebox generates a packet to notify the sender host of the heavy hitter to rate-limit. This is similar to NCF [22]. This design is referred to as BK-NIC. Algorithm 2 describes its pseudocode. BK-NIC is conceptually

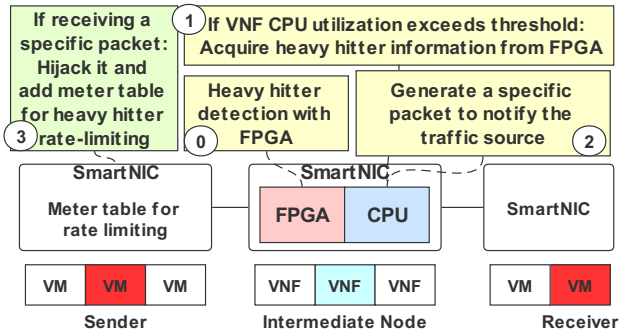


Fig. 5. BK-NIC architecture, using meter tables to rate-limit.

Algorithm 2: BK-NIC deployment.

```
1  Function Select_flows():
2      ...
3  Function Intermediate_NIC2():
4      if CPU utilization > Threshold then
5          FlowID_set = Select_flows()
6          for flow in FlowID_set do
7              Send a specific packet with 5-tuple to notify the traffic
                 source.
8  Function End_NIC1(packet):
9      if The packet is a specific packet then
10         Hijack it and add meter table with 5-tuple in the payload.
```

similar to NFVnice [3], which drops packets early at the beginning of the service chain through the NFV manager.

The per-flow backpressure is conducted with the following steps shown in Fig. 5: ⓪ heavy hitter detection runs with SmartNIC acceleration; ① if the CPU utilization of some forwarding instance exceeds the predefined threshold, the SmartNIC software will query FPGA to obtain the current heavy hitters and then select the ones to rate-limit based on packet count ratio; ② the intermediate node will then generate a specific packet to notify the traffic source of these heavy hitters (we can use one bit in the VXLAN header to specify that it is a notification packet and store the 5-tuple of the heavy hitter in its payload); ③ on receiving the notification packet, the SmartNIC at the traffic source will intercept the packet and add corresponding meter table entries for heavy hitter rate-limiting. The meter table makes sure that the transmission rate of the heavy hitter does not go beyond a certain limit, otherwise packets will be dropped. The meter table is implemented based on token buckets [21].

BK-NIC is independent of the guest protocol stack without needing any special support such as ECN, and thus works with both TCP and UDP heavy hitters. The notification packet may include the data rate to be limited to, and periodically update the meter table based on the current CPU load.

D. Backpressure to VM and SmartNIC

Although BK-NIC meets our design goals, the meter table rate-limiting may lead to excessive packet losses. Modern operating systems have some support for ECN, but they are usually shipped with ECN partially disabled. To reduce packet losses, we may leverage ECN with some 'tricks' to enable ECN for backpressuring, in addition to meter table rate-

TABLE I
BACKPRESSURE SCENARIOS.

Client tcp_ecn	Server tcp_ecn	Server to Client transmission	Client to Server transmission
1	1, 2	VM&NIC	VM&NIC
0, 2	1, 2	VM&NIC	NIC
1	0	NIC	VM&NIC
0, 2	0	NIC	NIC

limiting. This approach is referred to as BK-VM&NIC, as it backpressures to both VM and SmartNIC at the same time.

By analysing Linux kernel code [20], we find that as long as ECN state machine transforms to TCP_ECN_OK, we can backpressure using ECN. Table I shows the scenarios whether we can use ECN for backpressuring, in relation to client and server tcp_ecn values. VM&NIC means ECN can be used to backpressure to the source. NIC-only means ECN cannot work.

Full ECN support (Row 1 in Table I). As shown in § III-B, if tcp_ecn is 1 for the client and 1 or 2 for the server, the client will initiate ECN and the server will accept it. This enables backpressuring to both client and server VMs.

Deceiving ECN scenario 1 (Row 2 in Table I). When the tcp_ecn of the client and the server is set to 2 as default [9], a case in this scenario, the client will not initiate ECN, although the server would accept it if requested. ECN can not be established. However, we may 'trick' the server and move its state machine to the TCP_ECN_OK state by modifying the SYN segment from the client. The SmartNIC intercepts the SYN and sets the CWR and ECE fields to 1, and then forwards it to the server. After that, we can backpressure to the server VM using ECN, but not the client VM.

Deceiving ECN scenario 2 (Row 3 in Table I). If client tcp_ecn is 1 but server tcp_ecn is 0, the client will initiate ECN, but the server will ignore. Similarly, we may 'trick' the client by modifying the SYN-ACK segment and move its state machine to the TCP_ECN_OK state. We can then backpressure to the client VM, but not the server VM.

ECN not working (Row 4 in Table I). In the cases when both client and server are not interested in establishing ECN connectivity, we cannot backpressure to the VMs.

Algorithm 3: BK-VM&NIC deployment.

```
1  Function Intermediate_NIC1(packet):
2      ...
3  Function End_NIC2(packet):
4      if TCP_FLAGS = SYN then
5          Set CWR and ECE to 1.
6      if TCP_FLAGS = SYN-ACK then
7          Set ECE to 1.
8      if CE = 1 then
9          Set ECT to 1.
10         Set ECE of ACK packets with reversed 5-tuple to 1 until
             receiving a packet with CWR of 1.
11     if ECE = 1 then
12         Add meter table with reversed 5-tuple.
13         Set CWR of one packet with reversed 5-tuple to 1.
14     Forward the packet.
```

Implementation. Fig. 6 shows the backpressure pipelines

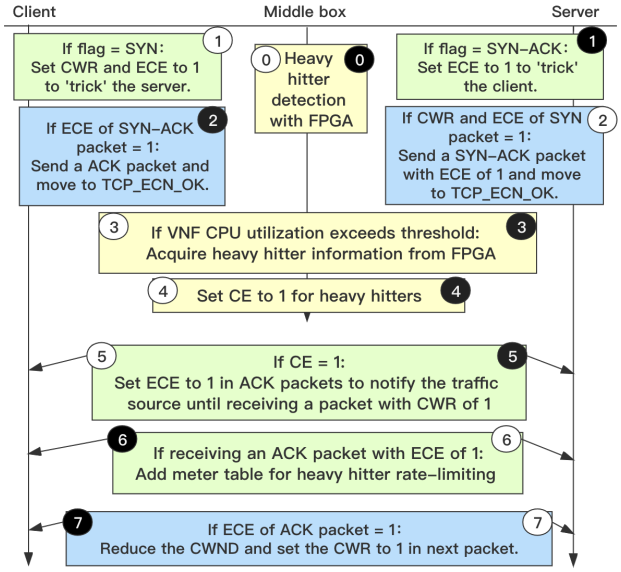


Fig. 6. BK-VM&NIC pipeline. It shows the steps to 'trick' the server (white label) and the client (black label), respectively. This will enable ECN for the transmission from the server or client. The trick only works if the host is willing to use ECN (Row 2 and 3 in Table I).

to 'trick' the client and the server VMs, with black and white labels, respectively. Using the pipeline to 'trick' the server VM as an example, here are the steps to enable ECN for backpressuring: ⓪ execute heavy hitter detection as with BK-NIC; ① when the SYN segment from the client VM is intercepted, the SmartNIC marks its CWR and ECE fields to 'trick' the server; ② on receiving the SYN with CWR and ECE of 1, the server VM sends a SYN-ACK segment with ECE of 1 and transforms its ECN state machine to TCP_ECN_OK state; ③ if the CPU utilization of some forwarding instance exceeds the predefined threshold, the SmartNIC software will query FPGA to obtain the current heavy hitters and select the ones to rate-limit; ④ then the intermediate node will mark CE for the packets of these flows to signal the impending congestion; ⑤ when the packet with CE of 1 is forwarded to the SmartNIC of the client VM, a match-action rule will be set to mark the ECE of the ACK packets until a packet with CWR of 1 is received; ⑥ on receiving the ACK with ECE of 1, the SmartNIC of the server VM adds a meter table for heavy hitter rate-limiting; ⑦ when the server VM receives the ACK with ECE of 1, it will reduce the CWND and mark the CWR of the next outgoing packet belonging to this flow. We follow the steps with black labels to 'trick' and realise backpressure to the client. The SmartNIC of the end-host implements all the green blocks in Fig 6 (line 3-14 in Algorithm 3). The SmartNIC of the intermediate node implements all the yellow blocks (line 9-13 in Algorithm 1). The blue blocks are implemented by the existing protocol stack.

Meter table working with ECN. Backpressuring to SmartNIC is often a backup method for backpressuring to VM's protocol stack. In the scenarios where the client or the server cannot be 'tricked', the meter table is the last resort for rate-limiting. Furthermore, while we do have a mechanism to set CE marking frequency so that the transmission rate of the

heavy hitter is maintained at a desired level, TCP congestion control tends to make the transmission rate fluctuate. The meter table can help stabilize the rate.

CE marking frequency. With high marking frequency at the middlebox, we may achieve lossless rate-limiting because of the low transmission rate imposed by TCP congestion control. But the throughput may suffer. With low frequency, the heavy hitter may still be able to maintain a high transmission rate, overloading the CPU. As a result, backpressuring to SmartNIC, that is, the meter table may kick in to further reduce the transmission rate by dropping packets. §V-A describes how to set a suitable frequency to achieve backpressure and high throughput with no or almost no packet losses.

E. Discussions

For TCP heavy hitters, BK-VM&NIC can achieve lossless backpressure by deceiving ECN and the careful setting of CE marking frequency. BK-VM&NIC can also regulate the transmission rate when it fluctuates due to the nature of TCP congestion control. In contrast, BK-NIC has to rely on meter tables to rate-limit by dropping potentially a large number of packets. BK-VM has limited applications due to its assumption of full ECN support. For UDP heavy hitters or when ECN is not available, BK-VM&NIC and BK-NIC work in the same way, rate-limiting by meter tables.

CE marking, packet dropping and *rwnd*. Our current implementation uses CE marking, emulating a fixed packet loss rate to control the transmission rate of the heavy hitters without dropping packets. CE marking frequency can be adjusted to achieve the desired transmission rate. If we relax the packet dropping policy, we may drop the packets of the heavy hitter at the middlebox to reduce its transmission rate. Alternatively, if we relax the non-invasive policy, we may set *rwnd* in the ACKs to a size that rate-limits the heavy hitter to the desired level without dropping packets [23]–[26].

Invasiveness. Our current approach keeps invasiveness at the minimum. We only play ECN 'tricks' if the VMs are willing to accept ECN. That is, the VMs have their tcp_ecn set to 1 or 2. Then, it is entirely up to their own congestion control to react to ECN marking. There are some interesting proposals [25], [26] that allow vendors to enforce an optimized congestion control (*e.g.*, DCTCP) at vSwitch or hypervisor without modifying the VM's protocol stack. The TCP segments and ACKs are intercepted and the relevant flags may be reset to enable ECN and emulate DCTCP. The *rwnd* in the ACKs is reset to reflect DCTCP congestion control (*cwnd*). This essentially forces the VM's TCP stack to emulate DCTCP or any congestion control schemes. This is not our goal to have this level of interference, given that we are only interested in some heavy-hitter flows at times. However, if such proposals get deployed we may choose a congestion control scheme favorable to backpressuring.

IV. Heavy Hitter Detection

To design a lightweight but accurate heavy hitter detection mechanism for rapid detection, there are some challenges.

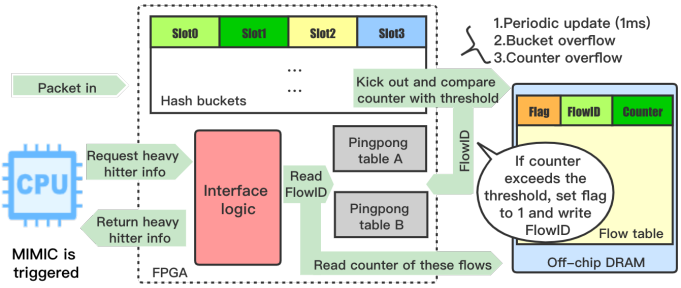


Fig. 7. FPGA architecture. A hierarchical memory desgin – on-chip hash buckets and off-chip flow table – originally for monitoring, auditing and accounting services. We extend the architecture by adding a lightweight heavy hitter pre-filtering procedure through the use of Pingpong tables.

On-chip SRAM + off-chip DRAM. Accurate detection relies on per-flow counting over a large time window to avoid misidentifying micro-bursts as heavy-hitter flows. The micro-bursts are tens of milliseconds in duration [27]. There are tens of millions of flows within a few seconds [28]. To store per-flow counting (∼100bits) for this sheer number of flows will require gigabits of memory. However, the on-chip SRAM is limited (*e.g.*, 345Mbits for Xilinx's vu9p [29]), and thus cannot accommodate per-flow counting on its own. The off-chip DRAM has the space. However, its limited bandwidth is an issue. If FPGA reports per-flow counting directly to DRAM every time when a packet is processed, the memory bandwidth will be insufficient. To this end we adopt a *two-level hierarchical memory design*, combining the on-chip SRAM with the off-chip DRAM (§ IV-A and § IV-B). This is already implemented and deployed in our production systems for monitoring, auditing and accounting services.

Bandwidth disparity between CPU and FPGA. The detection mechanism leverages this existing two-level memory design by adding a pre-filtering procedure (§ IV-C), which pre-selects a set of potential heavy hitters to be determined by the backpressure protocol (§ III-A). The backpressure protocol running on CPU gets this list of potential heavy hitters from pre-filtering running on FPGA through a heavy hitter query procedure (§ IV-D). However, there is a significant performance disparity between the two. The FPGA may *push* the heavy hitter information to the CPU. If the frequency of pushing is too high, the backpressure protocol running on CPU may be overwhelmed. If it is too low, the backpressure protocol may not have the latest heavy hitter information. To address this issue, we adopt a *pull* method. The backpressure protocol pulls the list of heavy hitters on demand.

A. FPGA-based hierarchical design

The per-flow counter updates for every single packet. Given the high throughput in the cloud (*e.g.*, tens of millions of PPS), this is a tremendous amount of work. If we process each packet by software with DPDK acceleration, the CPU core may be overloaded, resulting in packet losses. Our existing FPGA-based SmartNIC is a handy choice for this task.

Fig. 7 shows the FPGA-based hierarchical design taking advantage of the speed of the on-chip SRAM and the space and affordability of the off-chip DRAM, which is already deployed

in our production cloud. The design consists of (1) a CPU on the SmartNIC for triggering the heavy hitter query procedure when the CPU utilization spikes occur in the x86 server, (2) an FPGA for maintaining Hash Buckets to store the most recently counted flow states in a short-time window (*e.g.*, 1ms) and a Pingpong Table to store the FlowID of the current heavy hitters obtained through pre-filtering, and interfacing with the off-chip DRAM, and (3) an off-chip DRAM for maintaining the complete flow states in a long-time window (*e.g.*, 2s).

B. Hash buckets on FPGA

To deal with the limited on-chip memory, the current work has to trade off between detection accuracy and memory usage, resulting in some loss of flow information. Hash table keeps the state of each flow in an entry, which works well with per-flow counting for all flows, and has a low level of computation complexity. However, hash table does need a large amount of memory space. Our hierarchical design leverages the existing deployment of off-chip DRAM, making hash table a good fit. The trade-off between detection accuracy and memory usage is no longer an issue in our design.

The short-time flow table is stored in the on-chip memory based on *hash buckets* [30]. The hash function is only performed once each time when a new item is added to the bucket. A hash bucket has multiple slots to accommodate hash collisions. Each slot identifies a distinct FlowID and its counter. The off-chip DRAM helps keep the long-time flow table and handle the overflow of the hash buckets on FPGA.

FPGA overflow. When a packet arrives, FPGA identifies the bucket for its flow through hash function. If the flow is already present in the bucket, FPGA updates the corresponding flow counter. If the flow is not in the bucket, and there is an empty slot, we insert the new flow to the empty slot. However, if the bucket is full, and the flow is not found in the bucket, this results in *bucket overflow*. FPGA replaces the flow entry that has the minimum counter value in the bucket with the incoming flow. The evicted flow entry including its counter is copied into DRAM (Fig. 7). If the flow entry already exists in DRAM, the counter will be updated accordingly. In addition, the counter of each slot has an upper value. Too many packets of the same flow will result in *counter overflow*. The corresponding flow entry will be removed to DRAM. To avoid an excessive number of overflows, FPGA performs *periodic updates* every 1ms to evict all the flow states to DRAM.

C. Heavy hitter pre-filtering

The flow table in the DRAM keeps tens of millions of flows with gigabits of memory [28]. It is straightforward to identify the heavy hitters through sorting the counter, but there are some issues. Since the time required for sorting (approximately 10s) is much longer than that of periodic updates from FPGA (1ms), the counters constantly change during the sorting process, an inconsistent state leading to inaccurate heavy hitter detection. To tackle this problem, we introduce a heavy hitter pre-filtering mechanism. When the counters in the DRAM are updated by bucket and counter overflows or periodic updates, FPGA will write to the Pingpong table the FlowID for the flows whose counters exceed a threshold (Fig. 7). These flows are considered heavy hitters. Through pre-filtering, a large number of mice flows are filtered out and the flows that may be causing CPU overloading are left, which significantly reduces the time required for heavy hitter detection.

D. Heavy hitter query

In our design the backpressure protocol *pulls* the heavy hitter information from the FPGA in an on-demand manner. The CPU reads the pre-filtered heavy hitter information (§ IV-C) through the interface logic in FPGA (Fig. 7). That is, when CPU utilization in the x86 server exceeds the threshold, the CPU on the SmartNIC will issue a request to FPGA to get the current heavy hitters, which are stored in the Pingpong table through pre-filtering. To prevent blocking while reading and writing, there are two Pingpong tables, A and B, reading and writing to different tables. When FPGA reads table A for heavy hitter FlowID, pre-filtering writes to table B, and vice versa. When no reading occurs, pre-filtering writes FlowID randomly to one of the tables for load balancing. To prevent that the same FlowID gets written to Pingpong table multiple times, a 1-bit flag is used to indicate whether a flow has been recorded as a heavy hitter through pre-filtering. Upon the request from the CPU, FPGA generates a read command to both Pingpong tables A and B to get the FlowID of the heavy hitters, and reads the corresponding counter from the DRAM. FPGA then reports this information to CPU through interrupts. With this on-demand pulling, the CPU can obtain the heavy hitter information in a timely manner without wasting processing resources. Upon receiving the heavy hitters, backpressure is then performed on the selected flows according to the procedures in § III.

V. Experiment setting

We select one region to evaluate MIMIC. The guest OS is CentOS 7 3.10.0-514.21.1.el7.x86_64 with CUBIC TCP set as default. Each FPGA-based SmartNIC is configured with one 100Gbps NIC [31].

A. CE marking frequency

CE marking frequency has an impact on rate-limiting the selected TCP flows at the right level. However, different TCP variants have its own mechanism to adjust the congestion window. This potentially makes finding a universal CE marking frequency impossible. However, as the standard TCP works well in the networks with short RTTs, the newly proposed TCP variants such as CUBIC all aim (or are supposed to aim) to be fair, when competing with the standard TCP flows for bandwidth in such networks. This means the response functions of the TCP variants would exhibit a similar behavior as the standard TCP in these networks. We did some experiments to compare between the standard TCP and CUBIC with CE marking frequency within the region between 10^{-3} and 10^{-4}. They indeed exhibited the similar behavior. This gives the confidence to have a CE marking frequency for a given RTT

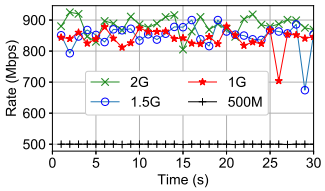


Fig. 8. CE marking frequency on data rate. Transmission rate is limited to around 850Mbps after CE marking regardless initial transmission rates.

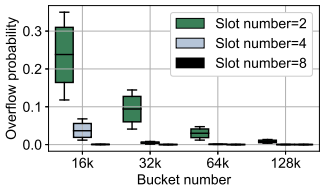


Fig. 9. Bucket and slot numbers on overflow. 32k buckets with 4 slots: a good trade-off between overflow probability and hardware resources.

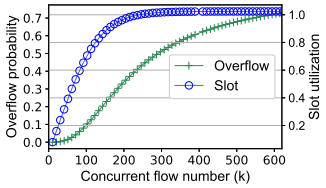


Fig. 10. No. of flows on overflow/slot utilization. For 40k flows (design assumption) low overflow probability is achieved with a good slot utilization.

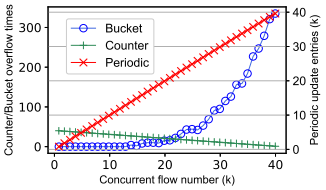


Fig. 11. No. of flows on overflow updates. The number of bucket overflows increases rapidly with the number of flows, but still well under 400.

in such networks, applicable to different TCP variants. The marking frequency can be adjusted to suit individual flows.

Based on our experience, 99% of the flows have a rate below 100Mbps and the CPU utilization of 1Gbps traffic is about 15%, which is commonly seen for the servers in data centers [32]. Considering the trade-off between CPU protection and throughput, the target transmission rate could be set to 800-900Mbps. We did some experiments with ECN enabled without meter tables in place. The TCP traffic is sent at different rates. The intermediate node marks CE at a constant frequency (1 over 2000 packets). Fig. 8 shows that with initial transmission rates at 2Gbps, 1.5Gbps and 1Gbps, all are limited to about 850Mbps while the transmission rate at 500Mbps is not affected. This observation is also supported by theory that with a deterministic loss model the transmission rate can be limited to a certain level. Therefore, if we want to limit the transmission rate to T Mbps, the marking frequency can be determined through response functions as well as experiments. To reinforce rate-limiting, the band rate of the meter table is also set to T Mbps to handle the busty nature of TCP as well as the scenarios where ECN is not supported. In the following experiments, the default band rate of the meter table is set to 850Mbps and the default marking frequency is set to 1 over 2,000 packets.

B. Hash bucket parameters

The number of hash buckets and its size are directly related to bucket overflow probabilities. Frequent overflows will lead to frequent updates between FPGA and DRAM causing unnecessary overhead. We need to trade off between the amount of required resources and overflow probabilities. A small number of buckets may cause frequent bucket overflows. With a large number of buckets, however, the allocated on-chip memory may have a low utilization rate. In addition, the number of slots in a bucket is proportional to the amount of hardware support (*e.g.*, LUTs, registers) used by FPGA for calculations and lookups. We want to minimize the numbers of buckets and slots with an acceptable overflow probability.

The number of buckets and slots are selected with performance analysis through simulations. Suppose our SmartNIC supports a maximum of 40Mpps packet processing rate, and thus a maximum of 40k packets arrive within 1ms. We simulated the hash bucket in software with 40k packets belonging to a random number of flows. In comparison to the use of production traffic, the simulation allows us to try out extreme scenarios: a large (small) number of small (large) flows. Fig. 9 shows the overflow probability with different bucket and slot numbers. It shows that with 2 slots the overflow probability is very high, making it unacceptable. With 8 slots the overflow probability is extremely low, however, the hardware support needed is the highest. The combination of 4 slots and 32k buckets appears to be a sweet spot. It has a low overflow probability with the best performance gain in terms of the use of on-chip memory and hardware support. Therefore, the bucket number is set to 32k with 4 slots in a bucket.

VI. Results

A. Hash bucket performance

The impact of concurrent flows. To validate our design on hash buckets, we perform a stress test. We adjust the number of concurrent flows up to 600k, which is well beyond our assumption of maximum 40k. The portion of heavy hitters is set to 1 over 1,000 flows. Fig. 10 shows that the number of slots in use increases with the number of concurrent flows and the slots are almost fully utilized with 200k flows. The overflow probability is increased from 0.01% to 72.65% as the number of flows grows from 10k to 600k. Note that there is a sweet region at the beginning of the curve where the overflow probability grows slowly while the slot utilization rate increases rapidly. This fits well with our assumption of maximum 40k flows with slot utilization at 30.9% and overflow probability at 0.82%. In addition, the overflow probability is about 30% even when the number of flows reaches 200k. This means that even at this scale much greater than the design assumption, our design can still reduce a large amount of overhead on the interactions between FPGA and DRAM.

Overflow. In order to show the proportions of the three interactive methods, we adjust the number of concurrent flows on the premise that the packet rate is 40k packets per ms. Fig. 11 shows that as the number of concurrent flows increases from 801 to 39957, the number of counter overflows decreases from 40 to 1 while the number of bucket overflows increases from 2 to 335. This is in line with our expectation that as the number of flows increases the bandwidth share for each flow is diluted, resulting in fewer counter overflows but more hash collisions and thus more bucket overflows. By the nature of bandwidth sharing, there will not be many heavy hitters. It is not surprising that the number of counter overflows is generally low, however the number of bucket overflows may explode as the number of flows increases. For a periodic update, as expected a similar number of entries are loaded to DRAM based on the number of concurrent flows in place

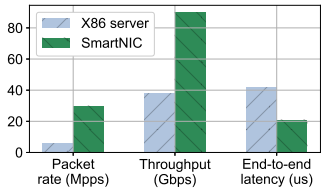


Fig. 12. Performance: SmartNIC vs x86 server. FPGA-based SmartNIC improves PPS performance 5 times for heavy hitter detection.

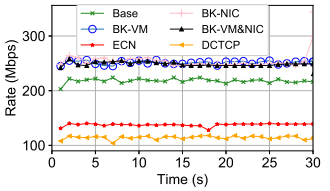
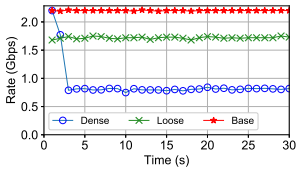
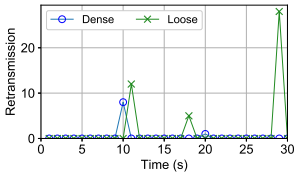


Fig. 13. Mice flow data rate. Mice flows benefit from rate-limiting heavy hitters, but get penalized by DCTCP for undifferentiated ECN marking.



(a) Heavy hitter data rate.



(b) Retransmissions.

Fig. 14. CE marking frequency on BK-VM. Loose marking cannot rate-limit heavy hitters to the desired level. Dense marking can achieve the goal.

(the number of entries is not exactly the same as the number of flows, because of overflows).

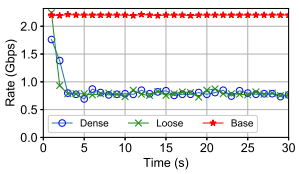
B. Heavy hitter detection ability

Fig. 12 compares the ability to detect heavy hitters between x86 server and SmartNIC with FPGA acceleration. SmartNIC increases packet processing rate by five times, throughput by more than two times and reduces end-to-end packet latency by more than half. This indicates that FPGA-accelerated SmartNIC is promising for cloud-scale heavy hitter detection. After testing in a region for one month, MIMIC detected several heavy hitters and performed backpressuring. There were no persistent packet losses observed in the testing.

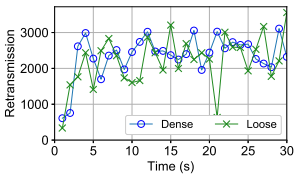
C. Rate-limiting performance

Mice flow. The per-flow backpressure should only be imposed to heavy hitters without penalizing the mice flows from the other tenants. Fig. 13 compares the data rate of mice flows between different schemes. The Base scheme has no rate-limiting applied. It shows that by rate-limiting heavy hitters the released bandwidth is taken up by the mice flows, resulting in improved data rate. In contrast, ECN (with CUBIC) and DCTCP do not differentiate between mice flows and heavy hitters for CE marking, causing a great deal of penalty on mice flows. DCTCP has its own ECE marking mechanism and the corresponding rate adjustment function. This has a greater impact on mice flows.

Heavy-hitter flow. Fig. 14, 15 and 16 show the rate-limiting performance between the three proposed schemes (namely, BK-VM, BK-NIC and BK-VM&NIC) with different CE marking and notification frequency, loose and dense marking. The loose marking is set to 1 over 5,000 packets. The dense marking is set to 1 over 2,000 packets, which gives the desired transmission rate (§ V-A). In the figures, the Base scheme has no rate-limiting applied. Fig. 14 shows that with dense marking BK-VM can rate-limit the selected heavy hitter to the desired transmission rate with almost no retransmissions.

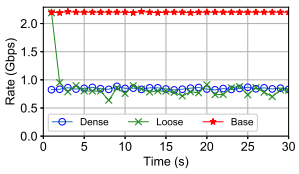


(a) Heavy hitter data rate.

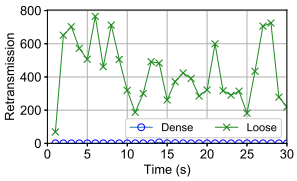


(b) Retransmissions.

Fig. 15. Notification frequency on BK-NIC. Meter tables can rate-limit heavy hitters to the level as desired. But excessive packet losses may occur.



(a) Heavy hitter data rate.



(b) Retransmissions.

Fig. 16. CE marking frequency on BK-VM&NIC. Regardless loose or dense marking the scheme can rate-limit to the desired level. Meter tables kicked in with loose marking, as it could not rate-limit to the desired level, causing a large number of packet losses. No packet losses with dense marking.

However, with loose marking the transmission rate is well above the desired level, persistently stressing on the CPU, although there are no many retransmissions. BK-NIC leverages the proposed notification packet to signal CPU overloading. Triggered by this packet, a meter table is then added to the SmartNIC on the sender host for rate-limiting the selected flows to the desired transmission rate. Fig. 15 shows that the meter table sets the transmission rate at the right level. However, there are a large number of retransmissions due to meter table's packet dropping mechanism. BK-VM&NIC is conducted not only on the VM but also the SmartNIC. That is, in addition to the rate-limiting mechanism through ECN on VM, there is also a meter table on the SmartNIC to further control the transmission rate if needed. Fig. 16 shows that the transmission rate is always limited to the right level regardless CE marking frequency. However, with the right marking frequency there were no retransmissions observed. With loose marking, there are some retransmissions but far below the level of BK-NIC. This is because the meter table on the SmartNIC does not have to drop as many packet due to the rate-limiting on VM.

BK-VM is similar to ConQuest [13] and NFVnice [3], requiring full ECN support. However, the later two use the standard CE marking while BK-VM can use the proposed mechanism for selecting CE marking frequency. BK-NIC is similar to NCF [22] as they both have a mechanism to generate a notification packet for backpressure. BK-NIC is similar to NFVnice [3] when its backpressure is done by a NFV manager, in the sense of dropping packets early.

D. x86 server CPU performance

RTT. Fig. 17 shows the query latency distribution between different schemes by pinging the corresponding CPU core. When there is no rate-limiting (Base), the queue in the middlebox is built up leading to unpredictable long latency. For BK-VM&NIC and BK-NIC, the latency is well controlled. The average latency for Base, BK-NIC and BK-VM&NIC are

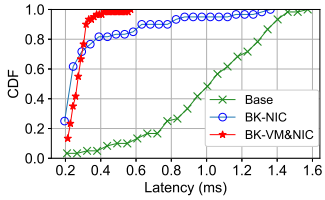


Fig. 17. x86 server query RTT. Query latency is well under control by rate-limiting heavy hitters.

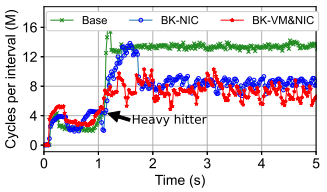


Fig. 18. x86 server CPU utilization with reaction time. Fast reaction in response to CPU overloading.

0.998ms, 0.402ms and 0.286ms, respectively. The "on/off" bursty nature of meter table tends to create micro queues. As a result, BK-NIC has a latency greater than BK-VM&NIC.

CPU utilization and reaction time. Fig. 18 shows the CPU cycles per 20ms interval for packet processing at the NFV node when CPU utilization exceeds the threshold. There are many mice flows as background traffic throughout the experiment. The black arrow points the time when the heavy hitter started. In the figure, our schemes significantly reduce the number of cycles. The reaction time is observed from CPU utilization exceeding the threshold to CPU utilization settling at the right level. The work in [33] implements heavy-hitter detection and backpressure in software, which takes nearly 1 minute to take effect. The reaction time of BK-NIC and BK-VM&NIC are 0.6s and 10ms, respectively, which is much faster than the software solutions. From our observation it takes about 1ms for the CPU to get the heavy hitter information, and about 0.3ms to get the flows selected and transmit the rate-limiting signal. It is then up to the protocol stack to receive, process and react to the signal. As a result, BK-VM&NIC reacts promptly and starts rate-limiting. BK-NIC has a longer reaction time because of meter table's rate-limiting mechanism. It takes the meter table some time to "warm up", observing the transmission rate before rate-limiting.

VII. Related work

Collaborative studies. A lot of efforts have been made to improve ECN marking at the intermediate nodes [34]–[37] and the end hosts [38]. These schemes are compatible with our use of ECN. Our CE marking can take advantage of these schemes when suitable. ACDCTCP [25] and vCC [26] allow vendors to enforce an optimized congestion control such as DCTCP in a transparent way. There is a high level of interference not suitable for MIMIC (§ III-E). However, if such systems get deployed, MIMIC may choose a congestion control scheme favorable for backpressuring.

On heavy-hitter detection, some systems [13], [14], [17]–[19], [22], [39]–[41] aim to achieve high detection accuracy with efficient use of on-chip memory. The limited on-chip memory cannot deal with the amount of cloud traffic, without losing flow information. With the FPGA-based hierarchical memory design, MIMIC does not have to trade off between detection accuracy and memory usage. MIMIC may use these schemes if memory becomes an issue.

Rate-limiting upon detection. ConQuest [13] detects heavy hitters based on a short-time window. This may yield very efficient use of memory at the cost of losing flow information and mistaking micro-bursts as heavy hitters. ConQuest uses the standard ECN. This means it will not work on UDP traffic or in the scenarios where ECN is disabled on either side of the hosts. ConQuest is similar to BK-VM in our design. NCF [22] identifies the heavy hitters with count-min sketch and sends NACKs to the traffic source of the top-k heavy hitters. Again, it is an excellent scheme for efficient use of memory which may result in the losses of flow information. It is not clear how rate-limiting is carried out in response to NACKs. NCF is similar to BK-NIC. NFVnice [3] may achieve backpressure for a service chain through a NFV manager or ECN when the chain spreads across multiple hosts. In the earlier case, a packet is dropped early at the beginning of the service chain to avoid wasting resources later on. Similarly, BK-NIC drops packets early at the source host. In the later case, NFVnice is similar to BK-VM, requiring full ECN support. However, while NFVnice can backpresure a particular flow, it does not try to identify the heavy hitters.

Alternative approach. The RDMA-based solutions [42]–[44] represent a different approach without relying on the existing protocol stack as long as the Ethernet fabric supports it. It requires a RDMA-capable NIC and its own stack to ensure high throughput, low latency and lossless transmissions. This is a direction worth investigation.

PicNIC [16] and EyeQ [15] monitor the receiving rate of each VM. When the rate exceeds a threshold, they will backpressure to the source VM with admission control. However, this cannot guarantee a congestion-free core. In fact, EyeQ assumes a congestion-free fabric for optimal bandwidth allocation. The admission control of EyeQ may lead to head-of-line (HoL) blocking and affect the mice flows by rate-limiting VM. To tackle the HoL issue and thus allow per-flow backpressuring, PicNIC needs to enable NAPI-TX and TSQ of the VM. NDP [45] does not need to perform congestion control based on the proposed stack modifications. These solutions are against our goal of being non-invasive, but worth further investigation. BwE [46] is a global, hierarchical bandwidth allocation infrastructure. The focus of BwE is on bandwidth allocation, not directly on backpressuring. However, BwE may help reduce the chance of congestion at the network core through accurate network modelling.

VIII. Conclusion

We build MIMIC, a cloud-scale flow backpressure system for production deployment. We extend our existing SmartNIC to facilitate heavy-hitter detection and per-flow backpressuring. MIMIC achieves non-invasive and controllable rate-limiting through the novel use of ECN and meter tables, together with TCP congestion control. For heavy hitter detection, MIMIC adopts a hierarchical memory design with pre-filtering, which results in no losses of flow information for improved detection accuracy and enables rapid detection. As a result, MIMIC can achieve a fast reaction time bringing down CPU utilization to the normal level without packet losses.

REFERENCES

[1] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, "When Cloud Storage Meets RDMA," in *USENIX NSDI 2021*, 2021, pp. 519–533.

[2] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, no. 8. Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[3] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "NFVnice: Dynamic backpressure and scheduling for NFV service chains," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 639–652, 2020.

[4] "The Current State of NFV Deployment and How It's Predicted to Change," https://www.vitria.com/wp-content/uploads/2019/01/The-Current-State-of-NFV-Deployment.pdf, 2019.

[5] P. Zheng, A. Narayanan, and Z.-L. Zhang, "A closer look at NFV execution models," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, 2019, pp. 85–91.

[6] S. D. Goglin and L. Cornett, "Flexible and extensible receive side scaling," Sep. 1 2009, uS Patent 7,584,286.

[7] T. Pan, N. Yu, C. Jia, J. Pi, L. Xu, Y. Qiao, Z. Li, K. Liu, J. Lu, J. Lu *et al.*, "Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proceedings of the 2021 ACM SIGCOMM Conference*, 2021, pp. 194–206.

[8] T. Barbette, G. P. Katsikas, G. Q. Maguire Jr, and D. Kostić, "RSS++ load and state-aware receive side scaling," in *Proceedings of the 2019 CONEXT Conference*, 2019, pp. 318–333.

[9] "Linux kernel," https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt, Accessed in 2022.

[10] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "RFC 8312: CUBIC for Fast Long-Distance Networks," https://datatracker.ietf.org/doc/html/rfc8312, 2018.

[11] K. Ramakrishnan, S. Floyd, and D. Black, "RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP," https://datatracker.ietf.org/doc/html/rfc3168, 2001.

[12] "OpenFlow Switch Specification," https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf, 2013.

[13] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. Wang, "Fine-grained queue measurement in the data plane," in *Proceedings of the 2019 CONEXT Conference*, 2019, pp. 15–29.

[14] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," in *Proceedings of the 2014 ACM SIGCOMM Conference*, 2014, pp. 419–430.

[15] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg, and C. Kim, "EyeQ: Practical network performance isolation at the edge," in *USENIX NSDI 2013*, 2013, pp. 297–311.

[16] P. Kumar, N. Dukkipati, N. Lewis, Y. Cui, Y. Wang, C. Li, V. Valancius, J. Adriaens, S. Gribble, N. Foster *et al.*, "PicNIC: predictable virtualized NIC," in *Proceedings of the 2019 ACM SIGCOMM Conference*, 2019, pp. 351–366.

[17] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the 2017 SOSR*, 2017, pp. 164–176.

[18] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A Better NetFlow for Data Centers," in *USENIX NSDI 2016*, 2016, pp. 311–324.

[19] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *USENIX NSDI 2013*, 2013, pp. 29–42.

[20] "Source code of tcp_output.c," https://elixir.bootlin.com/linux/v3.10.105/source/net/ipv4/tcp_output.c, Accessed in 2022.

[21] K. Chan, R. Sahita, S. Hahn, and K. McCloghrie, "RFC 3317: Differentiated Services Quality of Service Policy Information Base," https://datatracker.ietf.org/doc/html/rfc3317, 2003.

[22] A. Feldmann, B. Chandrasekaran, S. Fathalli, and E. N. Weyulu, "P4-enabled network-assisted congestion feedback: a case for NACKs," in *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019, pp. 1–7.

[23] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit Window Adaptation: A Method to Enhance TCP Performance," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, 2002.

[24] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad, "Receiver based management of low bandwidth access links," in *IEEE INFOCOM 2000*, 2000, pp. 245–254.

[25] K. He, E. Rozner, K. Agarwal, Y. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual congestion control enforcement for datacenter networks," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 244–257.

[26] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy, "Virtualized congestion control," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 230–243.

[27] "Understanding Overutilization and Microburst Behavior," https://www.gigamon.com/content/dam/resource-library/english/white-paper/wp-understanding-overutilization-and-microburst-behavior.pdf, 2021.

[28] B. Burres, "Intel's Hyperscale-Ready Infrastructure Processing Unit (IPU)," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–18.

[29] "Virtex UltraScale+," https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html, 2021.

[30] "CS3 Data Structures & Algorithms," https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/BucketHash.html, 2022.

[31] X. Zxt, Z. Zx, and J. Song, "High-density Multi-tenant Bare-metal Cloud with Memory Expansion SoC and Power Management," in *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE, 2020, pp. 1–18.

[32] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces," in *IEEE/ACM IWQoS 2019*. IEEE, 2019, pp. 1–10.

[33] J. Lu, T. Pan, S. He, M. Miao, G. Zhou, Y. Qi, B. Lyu, and S. Zhu, "A Two-Stage Heavy Hitter Detection System Based on CPU Spikes at Cloud-Scale Gateways," in *IEEE ICDCS 2021*. IEEE, 2021, pp. 348–358.

[34] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *USENIX NSDI 2016*, 2016, pp. 537–549.

[35] J. Zhang, W. Bai, and K. Chen, "Enabling ECN for datacenter networks with RTT variations," in *Proceedings of the 2019 CONEXT Conference*, 2019, pp. 233–245.

[36] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proceedings of the 2012 CONEXT Conference*, 2012, pp. 25–36.

[37] D. Shan and F. Ren, "Improving ECN marking scheme with micro-burst traffic in data center networks," in *IEEE INFOCOM 2017*, 2017, pp. 1–9.

[38] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the 2010 ACM SIGCOMM Conference*, 2010, pp. 63–74.

[39] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International conference on database theory*. Springer, 2005, pp. 398–412.

[40] A. Majidi, N. Jahanbakhsh, X. Gao, J. Zheng, and G. Chen, "DC-ECN: A machine-learning based dynamic threshold control scheme for ECN marking in DCN," *Computer Communications*, vol. 150, pp. 334–345, 2020.

[41] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[42] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster, and T. Edsall, "RoCC: robust congestion control for RDMA," in *Proceedings of the 2020 CONEXT Conference*, 2020, pp. 17–30.

[43] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "HPCC: high precision congestion control," in *Proceedings of the 2019 ACM SIGCOMM Conference*, 2019, pp. 44–58.

[44] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[45] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the 2017 ACM SIGCOMM Conference*, 2017, pp. 29–42.

[46] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila *et al.*, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *Proceedings of the 2015 ACM SIGCOMM Conference*, 2015, pp. 1–14.