IEICE TRANS. INF. & SYST., VOL.E102–D, NO.3 MARCH 2019

PAPER

Resilient Edge: A Scalable, Robust Network Function Backend

Yutaro HAYAKAWA[†a)], **Kenichi YASUKATA**[††b)], ***Nonmembers***, **Jin NAKAZAWA**[†c)], ***Member***, ***and*** **Michio HONDA**[†††d)], ***Nonmember***

SUMMARY Increasing hardware resources, such as multi-core and multi-socket CPUs, memory capacity and high-speed NICs, impose significant challenges on Network Function Virtualization (NFV) backends. They increase the potential numbers of per-server NFs or tenants, which requires a packet switching architecture that is not only scalable to large number of virtual ports, but also robust to attacks on the data plane. This is a real problem; a recent study has reported that Open vSwitch, a widely used software switch, had a buffer-overflow bug in its data plane that results the entire SDN domain to be hijacked by worms propagated in the network. In order to address this problem, we propose REdge. It scales to thousands of virtual ports or NFs (as opposed to hundreds in the current state-of-the art), and protect modular, flexible packet switching logic against various bugs, such as buffer overflow and other unexpected operations using static program checking. When 2048 NFs are active and packets are distributed to them based on the MAC or IP addresses, REdge achieves 3.16 Mpps or higher packet forwarding rates for 60 byte packets and achieves the wire rate for 1500 byte packets in the 25 Gbps link.

key words: *software switch, NFV, security, operating system*

1. Introduction

Emerging network infrastructure, including 5G and cloud networks, is designed to provide high-capacity and low-delay networks to users, and a fine-grained, rich set of in-network services, such as filtering, tunneling, mobility management, and video buffering [16], to each of them. SDN and NFV are expected to be key technologies in achieving flexible flow steering and traffic engineering, and network management at a scale.

Despite the relatively matured SDN and NFV techniques and evolution of programmable hardware including programmable switches and smart NICs, scalability and robustness of NFV backends typically implemented as software switches, are still challenging problems. First, the existing NFV backends scale only up to a few hundreds of ports where virtual network function (NF) instances, such as routers, firewalls, load balancers and proxies attach. This significantly limits the opportunity to utilize underlying hardware. The number of CPU cores per server is increasing; 10 Gbps NICs are ubiquitous and 100 Gbps ones are on the horizon. This phenomenon potentially accommodates much larger numbers of NFs. For example, on a 100 Gbps NIC, the server could run one thousand NFs when each NF processes packets at 100 Mbps.

Second, recent study demonstrated that even Open vSwitch, a widely-used production-quality software switch, contained a major bug in its data plane. It causes buffer overflow and allows the entire SDN system to be hijacked by worm propagated [29].

Even worse, these problems are mutually deteriorating; the more we serve users or tenants, the higher is the effect of the NFV backend vulnerability. Therefore, preventing NFV backends from software bug or resulting exploits is more important for consolidating a larger number of NFs. Further, since the NF and server backends are relatively low-level component of the entire software stack, their replacement or update requires significant effort and disrupts services, although their configuration is updated relatively frequently due to dynamic instantiation and migration of NFs.

In this paper we address these challenges by designing, implementing and evaluating Resilient Edge (REdge). REdge supports 4000 virtual ports to which NFs attach, and achieves 100 Gbps of throughput, while protecting itself by containing packet processing logic in the eBPF virtual machine. Protection, performance and scalability are generally conflicting goals. Protection degrades performance due to restricted execution environment and another layer of indirection; achieving high throughput and/or low latency tends to reduce scalability in the context of packet forwarding, due to the requirement for dedicating CPU cores to specific I/O threads.

We adopt the kernel-based data plane architecture, which is less common today in the context of high-performance software packet processing. In summary, this paper makes three contributions:

1. Architecture of safe, scalable NFV backend
2. Practical implementation based on mSwitch and eBPF
3. Evaluation of REdge's throughput and scalability

The remainder of this paper is organized as follows. Section 2 covers background to understand status quo in the literature, and clarifies problems. Section 3 shows the design and implementation of REdge. Section 4 evaluates

Manuscript received May 17, 2018.
Manuscript revised October 22, 2018.
Manuscript publicized December 4, 2018.

[†]The authors are with the Keio University, Fujisawa-shi, 252–0882 Japan.
[††]The author is with University of Liege, Belgium.
[†††]The author is with the NEC Labs Europe, Heidelberg, Germany.
a) E-mail: river@ht.sfc.keio.ac.jp
b) E-mail: kenichi.yasukata@student.uliege.be
c) E-mail: jin@ht.sfc.keio.ac.jp
d) E-mail: micchie.gml@gmail.com

DOI: 10.1587/transinf.2018EDP7176

REdge, and Sect. 5 discusses lessons learned from this work. Section 6 describes the rest of the related work. The paper concludes in Sect. 7.

2. Background

NFV backends are typically implemented using software switches that interconnect physical and virtual NICs, and forward packets between them by some logic, such as layer 2 learning, layer 3 routing and flow-based packet matching as with OpenFlow. Their architectures divide into the kernel and user-space approaches. This taxonomy may seem unfamiliar, but it actually exposes the trade-off between port scalability and packet forwarding throughput.

The kernel-based approaches [23] inherit scalability and generality offered by the kernel network stack, including interrupt handler, packet representation and device abstraction. However, over the last half decade, it has been argued that the kernel-based approaches cannot process packets at high rates required by 10 or 40 Gbps NICs. Researchers and practitioners have started implementing packet forwarding functionality in user-space using high performance packet I/O frameworks such as netmap and DPDK. These frameworks typically exploit zero copy, batching for system calls or device I/O, and lightweight packet buffer management.

User-space NF approaches appear to be successful, however, the same does not go for their backend. Individual NFs are designed to maximize their own performance, utilizing all the allocated resources. For example, they usually employ busy-polling for high performance packet I/O at the expense of high CPU usage. In contrast, NFV backends must minimize their own resource utilization to support as many NFs as possible. To address this problem, in 2015, mSwitch [11] has demonstrated to superior performance and scalability by merging the best aspects of both kernel- and user-space approaches. However, its design decisions still have scalability issues due to high memory usage and data copy, and a security problem described below.

In 2018, Thimmaraju et al. [29] report that existing software switches used as an NFV backend have vulnerability where attackers can hijack the entire network within minutes. They demonstrate this by injecting a malicious packet that causes a buffer overflow in the MPLS parser of the Open vSwitch data path [23]. They exploit the buffer overflow with ROP technique and propagate a worm to local and central controllers. Although this bug has been fixed, updating the operational systems requires significant efforts and affects many components of the system, such as orchestration tools, OS kernels, and many applications and VMs in the cloud. This distinctly implies that we should design the NFV backends to protect against vulnerability.

The primary challenge here is how to do so without sacrificing performance and scalability. Thimmaraju et al. [29] measure Open vSwitch performance with compiler-level protection enabled to quantify overheads of software switch protection. Unfortunately, their results indicate non-negligible overheads even for relatively low packet rates which are several hundreds of kilo packets per second. These rates are one order of magnitude lower than modern network bandwidth of 10 or 40 Gbps.

Table 1 Summary of performance characteristics of software switches: REdge supports all of high throughput, high port density and data plane security.

	Throughput	Port Density	Security
OVS (kernel) [1]	1 Mpps	1000s	×
OVS (DPDK) [11] and BESS [10]	> 10 Mpps	10s	×
Linux XDP ([31])	> 10 Mpps	×	eBPF
mSwitch [11]	> 10 Mpps	100s	×
REdge	> 10 Mpps	1000s	eBPF

Recent advances in the Linux kernel somewhat break this logjam, implementing a high-performance packet I/O engine called XDP and safe code execution environment for packet processing with Extended Berkeley Packet Filter (eBPF). However, it focuses on packet redirection to other NICs and filtering prior to moving received packets to the vast majority of the network stack. XDP cannot act as an NFV backend that serves hundreds or thousands of virtual ports, because of lack of parallelism on incast to a physical NIC transmitted by a large number of NFs and efficiency regarding delivering a batch of packets to multiple virtual ports. Further, XDP does not offer high-performance virtual NICs to which NFs attach.

Safe, dynamic packet filtering is not new and was proposed in 1996 [6]. However, it is important to note that this is being actively explored nowadays in response to the requirement for flexibility and performance. Table 1 summarizes the features of the existing software switches.

3. Design

In the previous section, we identified the problem with port scalability and security. We design REdge to solve these problems by achieving the following properties *simultaneously*:

- Scalability to thousands of NF instances
- Protection of data plane against software bugs and
- High packet rates of tens of millions of packets per second for small packets and high bit rates of tens of gigabits per second for large packets.

Figure 1 illustrates the REdge architecture. We implement it by extending mSwitch, a modular software switch that runs in the kernel. It attaches either physical NICs or virtual NICs to which VMs, containers or user-processes attach. Packet forwarding from a physical NIC is performed in the context of receive interrupt or dedicated polling thread, whereas that from a virtual NIC is performed in a system call context of the sender. This model is common in other kernel-based software switches, such as Open vSwitch and Linux bridge. This means that REdge architecture could be

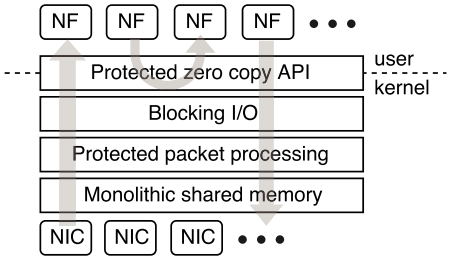


Fig. 1 REdge architecture.

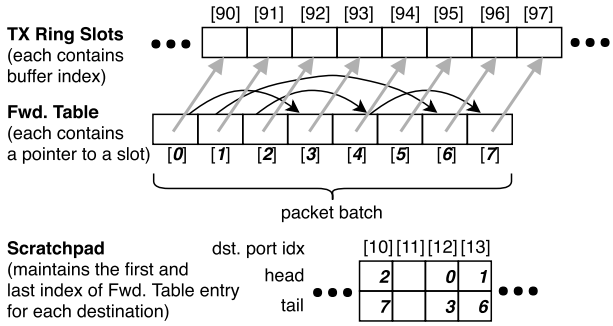


Fig. 2 REdge's packet forwarding data structures: REdge points buffer *slots* while mSwitch points packet buffers directly. In this figure packets at slots 2, 4 and 7 go to the destination port 10, those at slots 0 and 3 go to the port 12 and those at slots 1 and 6 go to the port 13.

applicable to these software switches. However, in this paper we base our solution on mSwitch because of its high performance and maturity (e.g., having been included in FreeBSD).

3.1 Scalable Packet Forwarding

NFV backends should be lightweight in terms of all of CPU utilization, memory usage and memory bus utilization, because NFs, including various middlebox implementations, also stress these system resources [9], depending on type of NFs. For example, header parsing are CPU intensive, and data copy (e.g., packet encapsulation) and tree traversal operations (e.g., IP prefix lookup) stress memory bandwidth and CPU caches. Therefore, in designing scalable NFV backends, it is important to minimize utilization of these resources within the backend and to allocate maximum of these resources to NFs. It should also be noted that certain resources, such as memory bandwidth and CPU caches are difficult to manage for isolation.

To allow NFs to achieve high performance and low resource utilization, it is useful to provide zero copy API to applications, but we should do so safely. Crash of an individual NF should not affect the rest of the system to minimize service disruption.

Monolithic shared memory: To avoid data copy end-to-end, REdge configures a memory pool shared by all the physical and virtual ports. This organization does not only enable zero copy across the input NIC, NF(s) and the output NIC, but also achieves better memory utilization, because we do not have to pre-allocate a portion of the memory to individual NFs. As a result, in the kernel, REdge always forwards packets between ports without data copy; NFs that run in user space or as VMs can also move packets between input and output ports without data copy, using the netmap API [26].

It should be noted that zero copy is not always fast; costs of data copy largely depends on NF type and hardware setup, such as the number of memory channels, memory clock speed and NUMA node locality. It also depends on CPU cache locality determined by the behavior of the individual NFs. For example, copying data to new location is expensive. Therefore, we design and implement REdge to be able to switch between data copy and zero copy forwarding on the fly.

Scalable zero copy forwarding table: REdge implements a new internal forwarding table in mSwitch. mSwitch sorts packets within a batch by output ports. This is because (virtual) device access involves locking of output port data structure and "trigger" action to notify the hardware or receiver NF of new packets, and grouping packets going to the same destination enables us to move multiple packets with a single locking and trigger action.

REdge's new table maintains pointers to ring *slots* whereas mSwitch maintains pointers to buffers. This enables zero copy packet forwarding by *swapping* buffers between source and destination slots. Figure 2 illustrates data structures leveraged in packet forwarding. This design adds a single step of conversion from slot to buffer on every packet, but avoiding data copy is essential to move large data efficiently.

In addition, since mSwitch only supports 255 switch ports, we should increase this limit. This increases the size of the data structure (Scratchpad in Fig. 2), which would exacerbate data locality. In Sect. 4, we demonstrate that this overhead is negligible.

3.2 Packet Processing Protection

We have observed that even production quality software data plane often contains a serious bug. It is unlikely that packet I/O engine contains serious bugs that result in exploits, because its input patterns are limited unless it concerns packet contents. However, packet processing code literally processes packet headers (and often payloads), which means we should assume almost infinite sets of inputs. It is clear that such code tend to contain buffer overflow or other bugs.

Generally, it is difficult to realize such safety without sacrificing performance, because it often requires runtime program checking which usually has unacceptable overhead for fast packet processing. To achieve better performance, packet processing code require to be checked statically.

REdge leverages eBPF implementation included in recent versions of the Linux kernel. eBPF itself is an independent 64bit register machine with a small subset of common CPU instruction architecture which can be Just-In-Time (JIT) compiled into native machine code. Linux ker-

nel forbids eBPF programs to contain loop to ensure that the program halts. Together with certain other restrictions such as limited numbers of instructions and small stacks, eBPF programs can be *verified* statically before execution. eBPF checks against instruction number limit, memory boundary violation, pointer leaking, invalid jump. eBPF thus protects the users against unexpected memory error or stack buffer overflow bugs. The bug reported in [29] can be avoided using this mechanism. Despite of these restrictions, eBPF has been proven to be sufficiently expressive to write packet processing code, together with extensions described below.

Since memory access from an eBPF program should be statically verifiable, persistent in-memory data structures such as routing table or filter rules also should be verifiable. Linux kernel provides in-memory key-value store which is called **map**. Map can be manipulated from userspace via file descriptor provided from `bpf(2)` system call, or from eBPF program via *call* instruction which can call restricted sets of external kernel functions. Such function calls are tracked using the verifier, and access to the map is always safe. Note that data structures implemented for map (e.g., hash table, trie) is considered to be safe.

Programming Interface

REdge's packet processing logic protected by eBPF can be written in familiar C syntax, implementing a *lookup* function that is invoked on every packet (Fig. 3) and returns an index of the destination switch port, or one of the two special values that indicate broadcast or drop (REDGE_BPF_BROADCAST and REDGE_BPF_DROP in Fig. 3, respectively). Further, eBPF programs written for Linux XDP can be easily modified for REdge.

Figure 3 shows the pseudo code of a lookup function that forwards packets based on the destination MAC address of input packet. Core logic implementation is similar to that of XDP eBPF. In the pseudo code, the `BPF_TABLE` function is used for describing a hash table. This function is not provided by REdge but the bcc [3] toolchain. Since REdge reuses those standard functionalities and utilities of bcc, the developers essentially only require modifying the XDP program to adapt the destination port specification.

```
// A hash table used via map.
BPF_TABLE("hash",
    struct mac, uint16_t, ft, 12288);

uint16_t
redge_lookup(struct pktbuf *pkt)
{
  uint8_t *data = (uint8_t *) pkt->data;
  uint8_t *data_end = (uint8_t *) pkt->data_end;

  struct eth *eth = (struct eth *)data;
  if ((uint8_t *)(eth + 1) > data_end) {
    return REDGE_BPF_DROP;
  }

  uint16_t *dport = ft.lookup(&eth->dst);
  if (dport == NULL) {
    // No entry found
    return REDGE_BPF_BROADCAST;
  }

  return *dport;
}
```

Fig. 3 Pseudo code of REdge eBPF.

3.3 Implementation

REdge is implemented by extending mSwitch. We modified 64 Lines-of-Code for scalability improvement and wrote 200 Lines-of-Code of mSwitch module for eBPF integration. Our implementation is potable. REdge runs in both Linux and FreeBSD. In Linux, REdge uses Linux native eBPF and XDP facility included in Linux with a thin glue. For FreeBSD, we ported uBPF to the kernel and integrated it with mSwitch.

4. Evaluation

We evaluate REdge to confirm the following hypothesis:

- REdge offers data plane protection with low overheads.
- REdge offers higher scalability than existing systems

4.1 Methodology

In addition to the complete REdge implementation, we use two variants of it to highlight effects of the different techniques: REdge-no-eBPF that does not use eBPF, REdge-copy that uses eBPF but copies data. In addition, we compare REdge and its variants against existing systems: mSwitch and XDP.

4.2 Experiment Setup

Hardware and OS: Unless otherwise stated, we conduct the experiment using a server equipped with a single Intel Xeon E5-2690v4 CPU clocked at 2.6 GHz, 64GB quad-channel memory and Intel XXV-710-DA2 25 Gbps dual-port NIC. This server installs the Linux kernel version 4.16. We disable all the offloading features, Turbo-boost and CPU C-states. We use another machine to generate and terminate the traffic that goes through these NIC ports.
Implementation of Packet Processing Logics: We use three packet processing logics that have different complexity and implement each of them in the normal C and eBPF. First one is "dummy" which statically forwards packets to an intended destination port, without parsing any packet header. Second one is "MAC", which forwards packets based on the destination MAC address. The third one is "5-tuple", which forwards packets based on a tuple of transport protocol, IP addresses and ports.

Both MAC and 5-tuple logics maintain entries in a hash table. In the logics written in eBPF, we use the map mechanism described in Sect. 3.2 to access the hash table. In the

Table 2 Baseline performance.

		Dummy		MAC		5-tuple	
	Topology	60B	768B	60B	768B	60B	768B [Mpps]
REdge	NIC-NF-NIC	15.34	3.94	12.36	3.94	9.78	3.94
	NIC-NIC	15.50	3.94	12.20	3.94	9.60	3.94
REdge-copy	NIC-NF-NIC	9.27	3.94	9.07	3.94	7.19	3.94
	NIC-NIC	16.90	3.94	10.80	3.94	8.83	3.94
REdge-no-eBPF	NIC-NF-NIC	16.54	3.94	12.43	3.90	10.17	3.94
	NIC-NIC	17.07	3.94	12.42	3.90	10.26	3.94
mSwitch	NIC-NF-NIC	10.77	3.94	11.26	3.89	7.67	3.94
	NIC-NIC	18.07	3.94	12.63	3.94	9.55	3.94
XDP	NIC-NF-NIC	2.19	1.57	2.46	1.57	2.42	1.57
	NIC-NIC	14.73	1.57	7.40	1.57	6.49	1.57

logics written in C, we reuse the eBPF map data structure for fair comparison. The 5-tuple logic has slightly higher complexity than the MAC logic, because the larger number of packet headers has to be parsed.

Note that any of our logics does not touch packet payload, so packet *processing* overhead is a function of not a bit rate but a packet rate. For REdge, 60B of packet size is always the worst case where packet rate becomes high and benefit of zero copy to move packets is diminished.

4.3 Data Plane Overheads

REdge is designed to achieve high port scalability and protection of packet processing code. We expect these two features to be achieved at the expense of certain performance penalty in comparison to not doing so. Therefore, we begin with measuring REdge's overheads by comparing REdge against existing systems and its partial building blocks.

Table 2 shows the packet forwarding throughput of various systems between the two NICs. We use two topologies: one runs NF (Fig. 4 (b)), and one does not (Fig. 4 (a)). The first topology is useful to observe end-to-end throughput, whereas the second one is useful to measure pure switching logic overheads. The machine activates a single CPU core and NIC ring in REdge, mSwitch or XDP. The NF runs on a separate CPU core. We allocate 2048 packet buffers for each of the NIC rings, and use a single CPU core to process the incoming packets. In the second topology, we use another dedicated CPU core for NF.

Overhead of scalability enhancement: Difference of throughput between mSwitch and REdge-no-eBPF demonstrates overhead caused by scalability improvement with REdge which includes the new forwarding table and zero copy packet forwarding (Sect. 3.1). For a simple packet forwarding scenario (Dummy, NIC-NIC, 60B), mSwitch achieves slightly higher rate, exhibiting 3 ns of additional per packet cost in REdge ($1/17.07 - 1/18.07$). However, when we employ more realistic, complex forwarding logic, this overhead becomes smaller, and REdge-no-eBPF even outperforms mSwitch on the 5-tuple-based forwarding logic.

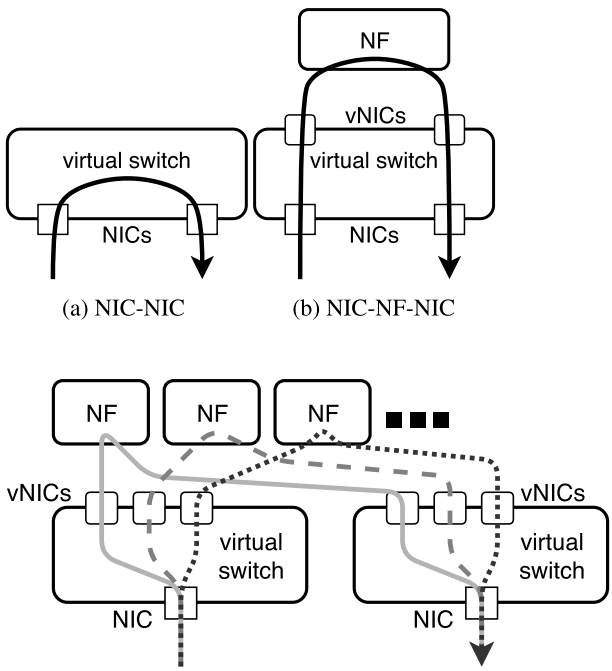


Fig. 4 Experiment topology: Virtual switch represents either mSwitch, REdge (or its variants) or XDP. vNIC represents a virtual port.

Moreover, in the presence of NF (NIC-NF-NIC), REdge-no-eBPF always performs better than mSwitch. REdge-no-eBPF saves three data copies in total, which include one from the input NIC to NF, one from input virtual port to output virtual port in NF and one from NF to the output NIC. These results imply that scalability enhancement made by REdge improves the performance at even small-scale NF deployment.

Comparison between REdge-copy and REdge indicates the effect of zero copy packet forwarding in the presence of eBPF and the new forwarding table. We observe similar performance characteristics to previous comparison between mSwitch and REdge-no-eBPF. In the dummy logic, the copy method is slightly faster, but when we use more complex MAC and 5-tuple logic, the zero copy method

demonstrates always higher performance. In the presence of NF, the zero copy method is always faster than the copy one. These results confirm that the positive effect of zero copy with REdge stays in the presence of packet processing protection and new forwarding table.

Overhead of packet processing protection: Performance difference between REdge-no-eBPF and REdge demonstrates overheads of protected packet processing logic. For 60B packets, depending on forwarding logic REdge exhibits 2–7 ns of additional per packet cost in the NIC-NIC case, and a similar number in the NIC-NF-NIC case. We believe these overheads are low, because per-packet time budget to forward minimum-sized packets at 25 Gbps line rate is 27 ns.

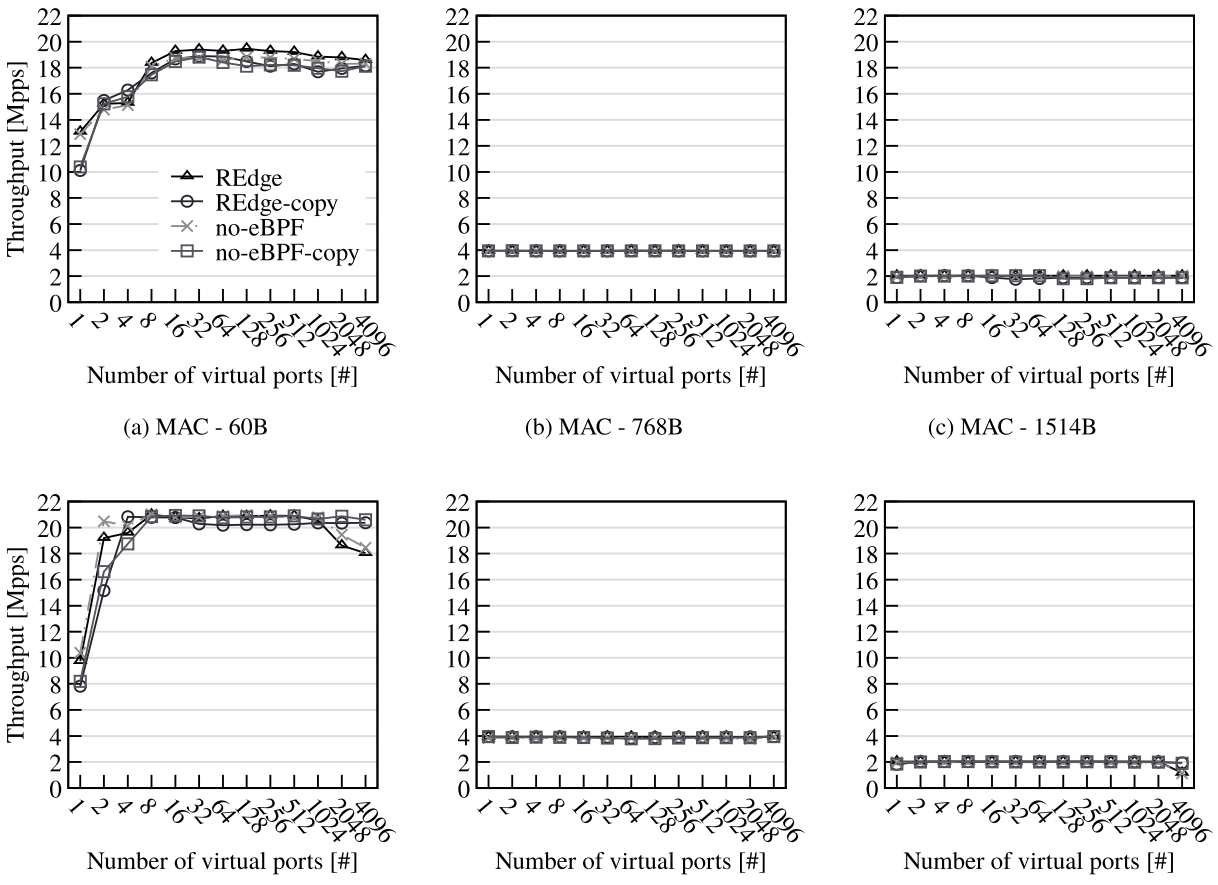
Overhead of both: Finally, performance difference between REdge and mSwitch indicates REdge's overheads that include both port scalability enhancement and packet processing protection. Observing the dummy, 60B, NIC-NIC case, we observe 9 ns of additional per-packet cost. However, we do not observe significant overhead in the MAC and 5-tuple cases. In the NIC-NF-NIC case, REdge always outperforms mSwitch. Based on these observations, we conclude that REdge offers high port scalability and packet processing protection practically at no cost.

Comparison to other systems: We are also interested in relative performance to Linux XDP which also shares the same objective with REdge in terms of protected packet processing. Unfortunately, owing to a device driver problem, XDP does not perform as expected. To elucidate its potential performance, we run the equivalent test but using Intel X540 10 Gbps NIC whose driver is more matured. We observed 14.73 Mpps with 60B packets, confirming that REdge achieves comparable performance even to non-scalable architecture. Note that very low performance of XDP's NIC-NF-NIC case is caused by `tap` interface used as the virtual port.

4.4 Scalability

We evaluate REdge's scalability by increasing the number of active virtual ports or NFs. We use the topology in Fig. 4 (c), and the MAC and 5-tuple forwarding logic. Each virtual port has a pair of TX and RX rings, and each ring has 512 packet buffers or slots except for the case of 60 B packets with 5-tuple logic which uses 1024 buffers to buffer the sufficient number of packets. Batch size is always 512 packets. All the NFs share the four CPU cores that are not utilized by the backend.

Figure 5 plots the throughput with increasing numbers of NFs for 60B, 768B and 1514B packets. We observe ap-



(a) MAC - 60B (b) MAC - 768B (c) MAC - 1514B

(d) FiveTuple - 60B (e) FiveTuple - 768B (f) FiveTuple - 1514B

Fig. 5 REdge scalability.

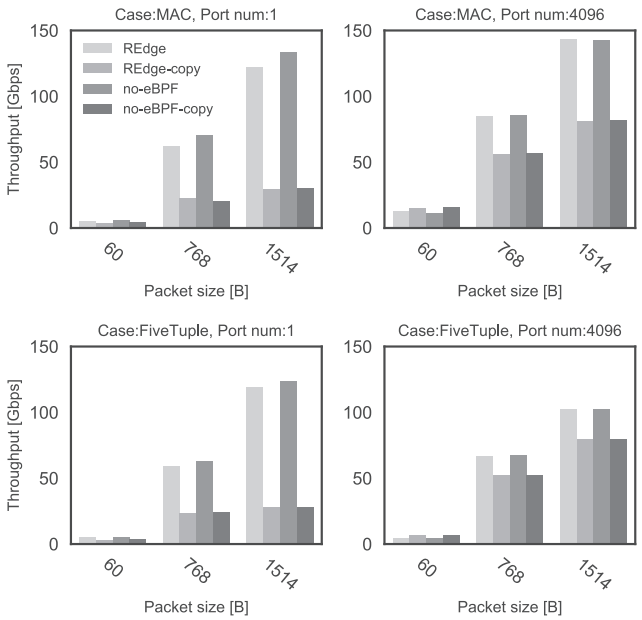


Fig. 6 Packet forwarding throughput with and without copying data.

proximately 18 Mpps with 60 B packets even with 4096 NFs being active, and almost the 25 Gbps line rate for 768 B and 1514 B packets.

Low throughput with 60 B packets and small numbers of virtual ports is owing to packet drop caused by shortage of buffers of virtual ports. We separately confirmed that this can be avoided when we use small batch sizes or more buffers at each NF to prevent packet drop.

We conclude that REdge can scale to a large number of virtual ports or NFs while achieving high throughput and protecting packet processing logic.

4.5 Effect of Zero Copy Forwarding

Despite of the high throughput and scalability with REdge, so far we have not observed a significant effect of avoiding data copy. This is because of our input packet rates that are at most 25 Gbps, and we expect zero copy becomes effective for higher input packet rates offered by higher speed NICs. We set out to test what happens in such a case using a pair of virtual ports instead of that of physical ports to inject packets to the switch at higher rates. Figure 6 plots throughputs with REdge and a version of REdge that does not employ eBPF in the presence of 1 or 4096 NFs. We observe a clear advantage of zero copy feature offered by REdge. For 768 and 1514 byte packets, the zero copy method improves throughput by 27–327%. For the minimum-sized packets, it degrades throughput by at most 28% due to the manipulation cost of packet buffer pointers. However, given the large gain, we consider this negative impact as acceptable.

5. Limitation

Restriction imposed by eBPF results in certain inconvenience in implementing packet processing logic used by NFV backend. Prohibition of recursion and loop, limited number of instruction (4096) and limited stack size (512 B) would prevent developers from implementing processing of nested or stacked protocol headers in a natural manner. However, NFV backends typically just route (and/or encapsulate/decapsulate) packets based on a small set of protocol headers. Therefore, we believe this limitation is acceptable for NFV backends.

Firewalls, such as Linux `iptables`, FreeBSD `ipfw` and OpenBSD `pf`, apply multiple filtering rules within a single function scope. Thus, the stack tends to grow. However, long rule chain would not be used *inside* NFV backends but *atop* them, forming service function chaining.

6. Related Work

There exist a large body of frameworks to deploy NFs. SoftFlow [13], OKO [5] and NetBricks [22] supports running function-based, stateful NFs within software switch data plane. These frameworks have the security problem that REdge addresses, and restrict NF implementation to function based, which significantly differs from the existing NFs and appears to have a deployment problem. BPFabric [14] protects its data plane, but has a problem with port scalability.

Approaches to run NFs in programmable hardware such as SmartNIC [4], [7], [15] are orthogonal to REdge.

Flurries [33] pools a large number of NFs and activates on-demand. However, the number of concurrent NFs is limited by the underlying NetVM [12], which is a software switch based on DPDK.

Click [20] is a framework to implement user- or kernel-space router data plane. FastClick [2] extends Click for fast user-space networking. Unikernels [17], [18] enables protected, lightweight applications including NFs, but it also has the port scalability at the backend. These systems are complementary to our work, and can run on top of REdge.

Cuckoo-Switch, Hyper-Switch [25], BESS [28] and VALE [27] are software switches, but they do not scale to a large number of ports or protect packet processing logic. HyperNF [32] is based on mSwitch but does not protect the packet processing logic. Ptnetmap [8], [19] is a fast VM-host interface, which REdge can use to offer fast packet I/O to NFs.

Programmable buffers (PBs) [16] temporarily buffer client traffic in a 5G core network. It extends mSwitch in a similar manner to us. However, it does not protect data plane, and measures port scalability to only up to 2000 ports.

In contrast to backend protection offered by REdge, Shieldbox [30], FastPaaS [35] and SafeBricks [24] are mechanisms to protect NFs. E2 [21] is a resource allocation framework for NFV, taking care of scale out and NF placement SDNFV [34] provisions NFV instances intelligently across multiple hosts. They are complementary to REdge.

Intel DPDK and netmap are the two major fast user-space packet I/O frameworks. However, they differ signif-

icantly. netmap runs in the kernel and exposes only packet buffers to user-space, and it provides safety and both blocking and non-blocking (busy-polling) packet I/O. Therefore, REdge provides the netmap API on top of it. Intel DPDK entirely runs in user-space, including device drivers. Since it is difficult to process interrupts in user-space, applications are required to perform busy polling.

7. Conclusion

In this paper we designed, implemented and evaluated REdge, a scalable, protected NFV backend architecture. This work has been motivated by ever increasing per-server hardware resources that potentially enable accommodating more NFs at a single server, and by the recent important observations in the literature that vulnerability in the software switch allows attackers to hijack the entire SDN domain. We optimize the trade-off between protection and performance, using a new architecture that protects high performance in-kernel software switch and minimizes resource utilization of it to allocate maximum resources to NFs as possible. We then achieved the single-server NFV system that forwards packets at 25 Gbps line rate in the presence of 4000 NFs.

As future work, we plan to apply REdge to a thin hypervisor of Xen. Owing to security concern, software switches have not been run within Xen hypervisor. However, strong protection achieved by REdge could enable it and provide opportunity for networking acceleration and flexibility.

Acknowledgments

This work was supported by National Institute of Information and Communications Technology.

References

[1] D. Ahern, N. Aleksandrov, and R. Prabhu, "Scaling the number of network interfaces on Linux," https://youtu.be/_t41_dlBO7Q, 2016.
[2] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," Proc. ACM/IEEE ANCS, 2015.
[3] bcc, https://github.com/iovisor/bcc
[4] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone, "Open packet processor: A programmable architecture for wire speed platform-independent stateful in-network processing," arXiv preprint arXiv:1605.01977, 2016.
[5] P. Chaignon, K. Lazri, J. François, T. Delmas, and O. Festor, "Oko: Extending Open vSwitch with stateful filters," Proc. ACM SOSR, ACM, 2018.
[6] D.R. Engler and M.F. Kaashoek, "DPF: Fast, flexible message demultiplexing using dynamic code generation," Proc. ACM SIGCOMM, 1996.
[7] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H.K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D.A. Maltz, and A. Greenberg, "Azure accelerated networking: SmartNICs in the public cloud," 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp.51–66, Renton, WA, USENIX Association, 2018.
[8] S. Garzarella, G. Lettieri, and L. Rizzo, "Virtual device passthrough for high speed VM networking," Proc. ACM/IEEE ANCS, 2015.
[9] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," ACM SIGCOMM CCR, vol.42, no.4, pp.1–12, 2012.
[10] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software NIC to augment hardware," Tech. Rep., UCB/EECS-2015-155, EECS Department, University of California, Berkeley, 2015.
[11] M. Honda, F. Huici, G. Lettieri, and L. Rizzo, "mSwitch: A highly-scalable, modular software switch," Proc. ACM SOSR, 2015.
[12] J. Hwang, K.K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.445–458, Seattle, WA, USENIX Association, 2014.
[13] E.J. Jackson, M. Walls, A. Panda, J. Pettit, B. Pfaff, J. Rajahalme, T. Koponen, and S. Shenker, "Softflow: A middlebox architecture for open vswitch," 2016 USENIX Annual Technical Conference (USENIX ATC 16), pp.15–28, Denver, CO, USENIX Association, 2016.
[14] S. Jouet and D.P. Pezaros, "BPFabric: Data plane programmability for software defined networks," Proc. ACM/IEEE ANCS, 2017.
[15] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M.M. Swift, and T. Lakshman, "UNO: Uuniflying host and smart NIC offload for flexible packet processing," Proc. ACM SoCC, 2017.
[16] Y. Lin, U.C. Kozat, J. Kaippallimalil, M. Moradi, A.C.K. Soong, and Z.M. Mao, "Pausing and resuming network flows using programmable buffers," Proc. ACM SOSR, 2018.
[17] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud, ACM SIGPLAN Notices, vol.48, no.4, pp.461–472, 2013.
[18] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam, J. Crowcroft, and I. Leslie, "Jitsu: Just-in-time sum-moning of unikernels," 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp.559–573, Oakland, CA, USENIX Association, 2015.
[19] V. Maffione, L. Rizzo, and G. Lettieri, "Flexible virtual machine networking using netmap passthrough," Proc. IEEE LANMAN, 2016.
[20] R. Morris, E. Kohler, J. Jannotti, and M.F. Kaashoek, "The click modular router," Proc. ACM SOSP, pp.217–231, 1999.
[21] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for NFV applications," Proc. ACM SOSP, pp.121–136, 2015.
[22] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," OSDI, 2016
[23] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp.117–130, Oakland, CA, USENIX Association, 2015.
[24] R. Poddar, C. Lan, R.A. Popa, and S. Ratnasamy, "Safebricks: Shielding network functions in the cloud," 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp.201–216, Renton, WA, USENIX Association, 2018.
[25] K.K. Ram, A.L. Cox, M. Chadha, and S. Rixner, "Hyper-switch: A scalable software virtual switching architecture," Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), pp.13–24, San Jose, CA, USENIX, 2013.
[26] L. Rizzo, "netmap: A novel framework for fast packet i/o," 2012 USENIX Annual Technical Conference (USENIX ATC 12), pp.101–112, Boston, MA, USENIX Association, 2012.
[27] L. Rizzo and G. Lettieri, "VALE, A switched ethernet for virtual machines," Proc. ACM CoNEXT, pp.61–72, 2012.
[28] S. Han, "BESS," https://github.com/NetSys/bess/wiki, 2015.

[29] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Taking control of SDN-based cloud systems via the data plane," Proc. ACM SOSR, 2018.
[30] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "ShieldBox: Secure middleboxes using shielded execution," Proc. ACM SOSR, 2018.
[31] XDP, "The IO visor project," https://www.iovisor.org/
[32] K. Yasukata, F. Huici, V. Maffione, G. Lettieri, and M. Honda, "HyperNF: Building a high performance, high utilization and fair NFV platform," Proc. ACM SoCC, pp.157–169, 2017.
[33] W. Zhang, J. Hwang, S. Rajagopalan, K.K. Ramakrishnan, and T. Wood, "Flurries: Countless fine-grained NFs for flexible per-flow customization," Proc. ACM CoNEXT, pp.3–17, 2016.
[34] W. Zhang, G. Liu, A. Mohammadkhan, J. Hwang, K.K. Ramakrishnan, and T. Wood, "SDNFV: Flexible and dynamic software defined control of an application- and flow-aware data plane," ACM Middleware, 2016.
[35] W. Zhang, A. Sharma, K. Joshi, and T. Wood, "Hardware-assisted isolation in a multi-tenant function-based dataplane," Proc. ACM SOSR, 2018.

Yutaro Hayakawa is a graduate student at Graduate School of Media and Governance, Keio University. His research interests include operating systems and networks, especially for SDN (Software Defined Networks) and NFV (Network Function Virtualization) systems that include programmable data plane. He received his bachelor degree in 2016 at Keio University. In 2015 summer, he was an intern at NetApp in Munich, and published the resulting paper at ACM SoCC 2017. In 2016 summer, he was an intern at NEC Laboratories Europe in Heidelberg, Germany.

Kenichi Yasukata is a Ph.D. student at Department of Electrical Engineering and Computer Science at University of Liege, Belgium. He received his Bachelor's degree (Faculty of Environment and Information Studies) and Master's degree (Graduate School of Media and Governance) from Keio University, Japan. His research interests include operating systems, networking and virtualization. He has been working on high-performance networked systems.

Jin Nakazawa is an associate professor at Faculty of Environment and Information Studies at Keio University, Japan. He has received his Bachelor's degree (1998) in Faculty of Policy Management, Master's degree (2000) and Doctor of Philosophy's degree (2003) in Media and Governence from Keio University. His research interest includes middleware systems, distributed systems, ubiquitous computing systems, and life logging. He is a member of IEEE, IEICE, and IPSJ.

Michio Honda is a senior researcher at NEC Laboratories Europe in Heidelberg. Before that, he was a software engineer at NetApp in Munich. He received his Ph.D. degree in 2012 at Keio University in Japan. He has worked on transport protocols, middleboxes, network stacks, software switch. He has published in top-tier venues including ACM IMC, HotNets, CCR, SOSR and SoCC, and USENIX NSDI and ATC. He received IRTF/ISOC Applied Networking Research Prize in 2011 for his IMC paper, and best paper award at ACM SOSR '15. He has also been a contributor to the netmap framework. He has also served on technical program committee at ACM/IEEE SC 2019, USENIX ATC 2017 and 2018, ACM SOSR 2018 and many other conferences.