Multimedia Content Delivery with NFV: From the Energy Perspective

Silvery Fu, Jiangchuan Liu and Wenwu Zhu

Abstract—In today's Internet, multimedia traffic accounts for the largest share of all traffic, highlighted by its volume, variety, multicast nature, and QoS constraints. Downstream towards consumers, multimedia traffic may traverse through middleboxes, undergoing additional data processing imposed by content providers and/or distributors. With the advent of Network Function Virtualization (NFV), middleboxes are progressively embedded in off-the-shelf, general-purpose servers. Despite the benefits, NFV may incur an undue amount of energy consumption when carrying out high packet forwarding performance. In this article, we investigate how switching to state-of-the-art NFV products for multimedia content delivery can result in significant energy costs. We identify the energy inefficiency issue in the NFV dataplane which can be exacerbated if not handle properly. We outline a power management framework that considers characteristics of multimedia traffic and exploits CPU frequency scaling to save energy.

Multimedia traffic, in particular video traffic, accounts for the largest share of all traffic in today's Internet. By 2020, an estimation of 82% of consumer Internet traffic will be attributed to video streaming according to Cisco [1]. Multimedia traffic is highlighted by its volume, variety, multicast nature and additional QoS constraints. Downstream towards consumers, multimedia traffic often traverse through middleboxes (i.e., *network functions*; in this article, we use terms middlebox, network function, and network appliance interchangeably), such as WAN (Wide Area Network) optimizers, transcoders, content caches, NATs (Network Address Translator) and traffic shapers, undergoing additional data processing imposed by content providers and/or distributors.

Traditionally, middleboxes are implemented as dedicated, vendor-specific hardware, an approach that leads to escalated management costs and an inefficient use of infrastructural resources [2]. Network Function Virtualization (NFV) is an on-going movement led by global network operators that aims to migrate network functions from dedicated hardware to off-the-shelf, general-purpose servers. Potential NFV operators include cloud providers, network providers, and Telco CDNs (Content Distribution Network); transitioning to NFV can help them alleviate the capital expenditures (CAPEX) and operating expenses (OPEX). For instance, an attractive deployment target for NFV is carrier network edge or their Central Offices (see: http://opencord.org/).

While there are comprehensive works surrounding NFV's performance and architectural design [2] [3] [4], its energy cost has rarely been studied. Given the extensive presence of middleboxes, energy cost (as a major contributor of OPEX) is likely to become one of the deciding factors in the operators' adoption of NFV. In this article, we show that NFV powered middleboxes are prone to high energy overhead when delivering multimedia content. In particular, we find there exists energy inefficiency in the data forwarding component of major NFV platforms. We demonstrated this inefficiency is inherent to its design that excessively uses CPU cycles to attain high performance. Since multimedia traffic is persistent throughout the streaming session and usually impose additional QoS constraints, existing energy saving methods may not function well. Based on these observations, we propose an outline of a power management framework for NFV-based multimedia content delivery. We show that CPU frequency scaling can achieve promising energy savings without compromising the performance of multimedia applications.

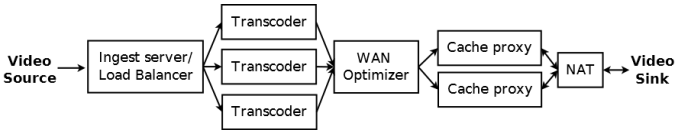


Fig. 1: A holistic view of middlebox traversal of a video stream.

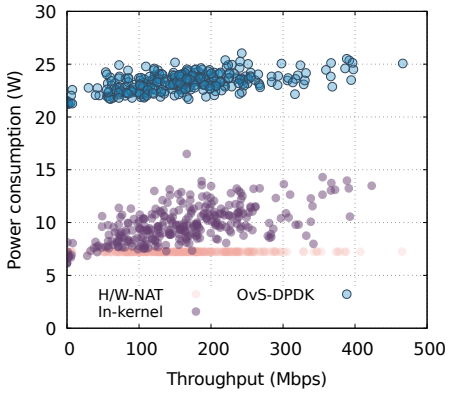


Fig. 2: Throughput and power consumption of a NAT box when multimedia traffic is passing through. Three middlebox deployments in comparison: hardware NAT, virtual NAT with Linux in-kernel bridge, and virtual NAT with Open vSwitch and DPDK forwarding engine.

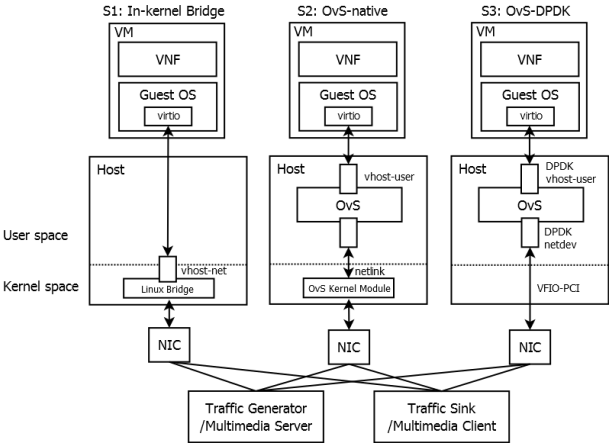


Fig. 3: The NFV testbed consists of three dataplane setups: off-the-box in-kernel bridge, Open vSwitch with native and DPDK forwarding engines, and two servers to inject traffic. Each virtual machine (VM) emulates a physical machine, providing functionalities to run an OS and hence network functions.

1 The Energy Cost of Transitioning to NFV-based Multimedia Content Delivery

NFV is often considered as an extension of cloud computing to the networking domain. As a multimedia content provider or distributor, however, transitioning to NFV may not be as easy as deploying a cloud application on its datacenter. Multimedia traffic often traverses through a series of middleboxes, as described in Figure 1. This simplified example captures how popular live broadcasting service providers, e.g., Twitch.tv, collect content (video broadcasters send streams to the ingest server), process content (video transcoding) and deliver content (via CDN) [5]. Two observations can be made here: first, there are different types of middleboxes [6] along the path, and they may possess various runtime characteristics such as resource usage and energy cost. Second, middleboxes are likely to be owned by different operators. When transitioning to NFV, the operators need to reevaluate their power budgets with the knowledge of energy costs for different middleboxes deployed from place to place.

To better understand the energy cost, we measured the power and bandwidth consumption of a NAT box with multimedia traffic is passing through. A typical NAT box modifies the IP header of in-transit packets to remap one IP address space into another. We chose the NAT box for its prevalence in today's Internet as well as its simplicity: being a network function with little application-layer processing, which provides a cleaner baseline for our future measurements. We set up an ASUS RT-AC68U router with NAT acceleration turned on for the hardware NAT, and the Linux `iptables` utility for the virtual NAT. We used the `ffserver` (https://trac.ffmpeg.org/wiki/ffserver) to set up a multimedia broadcasting server and `ffplay` (https://ffmpeg.org/ffplay.html) as the client. The sample multimedia traffic consists of a 1080p video stream (H.264 encoded, 3 Mbps bitrate), two audio streams (320 kbps and 160 kbps bitrate) and meta data. We devised a script that consecutively creates 300 sessions (starting from 0 session and every 10 seconds 10 more sessions are added) in 5 minutes. More details on the testbed setup and data collection can be found in Section 2.

We plot the results in Figure 2. As shown, the hardware NAT, sitting at the bottom, sustains the power consumption of around 7.2 watts regardless of the network load. Sitting in the middle is the virtual NAT implemented with Linux in-kernel bridge, whose power consumption is roughly in proportion to the load. Although the virtual NAT achieves lower power consumption than the hardware NAT when idle (6.16 W vs. 7.18 W), its power consumption is considerably higher (average 9.84 W vs. 7.20 W) with the highest readings doubling the power cost. Finally, the virtual NAT implemented with Open vSwitch and Intel DPDK forwarding engine, a popular module used in NFV platforms [3] [4], consumes significantly more energy than the other two setups (average 23.24 W), tripling the cost of hardware NAT. Moreover, *it consumes around 22 watts even when there is no traffic passing through*. The results indicate that energy cost may indeed become a concern for multimedia content provider and distributors when they transition to NFV.

2 Background and Testbed

By drawing a comparison with the hardware middlebox, one can see that the NFV products may greatly raise the energy costs when delivering multimedia content. To track the cause, we built a single-host NFV testbed as summarized in Figure 3. We use the following terms to refer to each component of the testbed:

- **Virtual Network Function (VNF)** A network function implemented by software and embedded in commodity server.
- **Virtualization Layer** An isolated runtime environment for VNFs, e.g., virtual machines (VM) or application containers; we chose the former and used Linux qemu-kvm utilities to create VMs.
- **Dataplane** The term originates from Software-defined Networking (SDN), which addresses the separation of network control plane (deciding how packets are routed) and the data plane (forwarding and/or processing packets). In some literature a dataplane may refer to any network component that operates on the traffic, including a VNF. In this article, we find it convenient use the term to describe the combined module of a vSwitch and its forwarding engine *only*.
- **Virtual Switch (vSwitch)** A software program that facilitates network communications among VMs. Virtual switches often support SDN flow rule interface (e.g., OpenFlow), and can be thereby more versatile than a software bridge (e.g., Linux bridge).
- **Forwarding Engine** A set of drivers and libraries that perform packet transmission between a physical NIC and the virtual NIC (and thus to the VNF).

We set up a midrange server with a 3.4GHz Intel i7 quad-core CPU and an Intel I350 Gigabit Ethernet Network Interface Card (NIC) as the host machine. We installed an additional Ethernet card to have an "out-of-band" control of the server, as in some of our configurations, the access of NIC will bypass the host's kernel entirely and cause the host to loss its IP stack functionality. We set up two physical

servers as the traffic generator and receiver. The three physical machines are connected via a Netgear GS116NA Gigabit switch. Finally, we created the following dataplane setups (as shown in Figure 3):

Linux in-kernel Bridge The bridge is available in most of mainline Linux distributions, making it an ideal out-of-the-box solution for setting up an NFV platform.

Open vSwitch Open vSwitch (OvS) is one of the most widely deployed software switch in the market. It features advanced flow caching, fast packet classification supports, and compatibility with OpenFlow. In addition to its native kernel forwarding engine, OvS also supports third-party forwarding engines such as Intel DPDK (http://dpdk.org/), targeting to better line-rate performance. We built two versions of OvS from the source (https://github.com/openvswitch/ovs), one with the native forwarding engine (OvS-native) and the other with DPDK (OvS-DPDK).

Having two OvS versions with different allows us to track down the energy consumption in vSwitch and forwarding engine separately. Across all setups, we configured `virtio` (http://www.linux-kvm.org/page/Virtio) as the datapath between the host and VM.

3 Tracking the Energy Inefficiency

To begin with, we consider the following characteristics of multimedia traffic:

- **Continuous** Multimedia traffic are persistent, continuous and sizable flows that occupy the link bandwidth during a time period.
- **QoS constraints** Multimedia applications are *user-facing*. They impose additional and possibly time-varying QoS requirements such as end-to-end delay, bandwidth, and packet loss rate.

These characteristics are likely to contribute to the raised NFV power consumption and affect the efficacy of power management methods. For instance, due to the continuity of multimedia traffic, it is difficult to spot an idle time of the traversed middleboxes. Hence it may become prohibitive to use sleep modes and deprovisioning unused middleboxes to achieve energy savings. Due to the QoS constraints, power management frameworks must also take into account the application performance when making power tuning decisions, a problem that is unfortunately non-trivial [7]; whereas, for those background, non-user facing applications, the middleboxes and network traffic can be better scheduled to achieve energy savings with fewer concerns. In addition, multimedia traffic can trigger intensive computations at the middleboxes and lead to high energy consumption. In the previous experiment we evaluated the NAT box, a network function that does not touch the traffic payload (i.e. the content) and involves little application-level processing; while there exist middleboxes that are computational-intensive and pervasive such as the *transcoding boxes* and *content distribution boxes* [5] [6].

In Figure 4, we depict the experiment of multimedia traffic traversing a transcoding box. We turned on the Linux `on-demand power governor`, a power saving module available and used in mainline Linux distributions. It tunes the CPU frequency based on the CPU utilization to conserve energy. As compared to the NAT box traversal, the energy

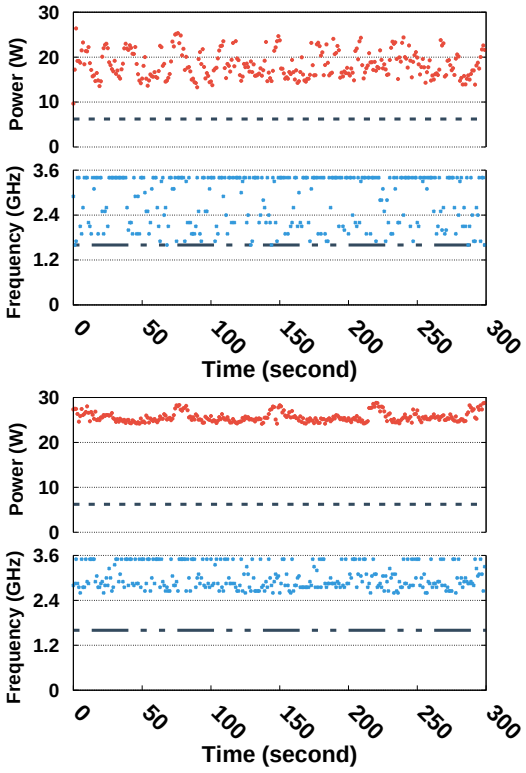


Fig. 4: Power consumption and CPU frequency measurements when multimedia traffic traverse transcoding box on in-kernel bridge (top) and OvS-DPDK (bottom), with the common power manager of OS turned on. The baseline power consumption (6.22 W) is the host server running without any VNF or dataplane; the baseline CPU frequency is the lowest supported P-state frequency.

cost is raised considerably: when the in-kernel bridge is used, the average power consumption is now close to 20 watts, doubled from the previous case ($\approx$ 10 W); for OvS-DPDK, it is now close to a staggering 30 watts. Besides, the majority of frequency is set to the maximum frequency (3.6 GHz) for both cases, indicating that the power manager may have limited influence on the overall energy savings (we offer discussion over this issue in Section 4.2). Moreover, we can see that the combined effect of dataplane and application-level processing on frequency and power consumption. The CPU frequency with OvS-DPDK stays above 2.4 GHz, hovering around 3.0 GHz and close to the maximum frequency. As a result, the consumed power is consistently higher than 20 watts, more than four times of the baseline.

3.1 Benchmarking the Dataplane

The experiment with real-world multimedia traffic indicates that NFV may introduce high energy overhead for content providers and distributors. They also hint us on potential energy inefficiency: the dataplane. Supporting the same VNF, the OvS-DPDK dataplane consumes much more power than the in-kernel counterpart (Figure 2). Yet it is not clear whether the vSwitch or forwarding engine is to blame.

We ran the following experiments to investigate these issues. We used `iperf` (iperf: https://iperf.fr/) network benchmark to generate traffic between the source and the

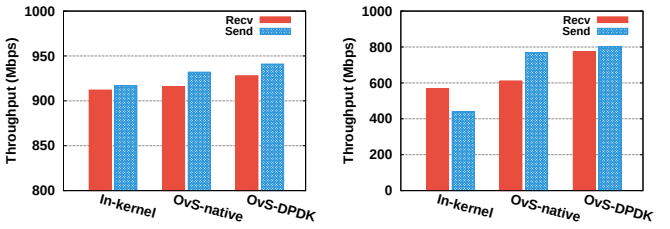
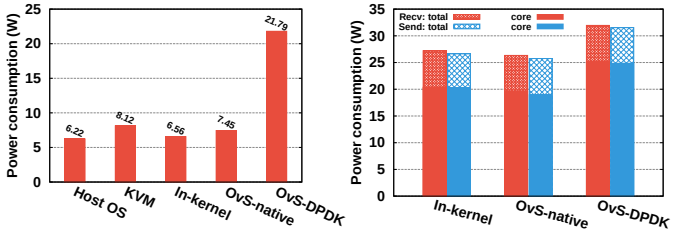


Fig. 5: Performance comparison of three dataplane setups under different packet rates: MTU 1500 (Left) and MTU 500 (Right). Note the difference in the Y-axis scale.



(a) Standby (b) Busy

Fig. 6: Power consumption comparisons of three dataplane setups in standby (without traffic) and busy (with traffic) scenarios and baseline measurements for host OS and VM. MTU is set to 500.

sink. Using the network benchmark allows the traffic generator to conveniently saturate the link capacity and put the dataplane under heavy load. It also allows us to evaluate the send and receive behavior separately. Besides, we were interested in the effect of different packet rates. We adjusted the packet rate by changing the MTU at the traffic generator.

We used Intel RAPL (Running Average Power Limit) (https://01.org/rapl-power-meter) to measure power consumption. RAPL provides both the holistic power readings (cores, caches, and memory controller) as well as core-only power readings, allowing us to determine the energy contributions of core and non-core components.

We present the network performance results with three dataplane setups in Figure 5. At the default 1500-byte MTU, both OvS-native and OvS-DPDK achieve higher throughput in TCP sending and receiving than the out-of-the-box in-kernel bridge, although the performance difference is only marginal ($< 5\%$). When the MTU is 500, with roughly three times more packets to process, the performance of three setups now differs prominently. The TCP send throughput of in-kernel bridge plummets to 441 Mbps, less than half of the original performance. Although the other two switches also suffer from performance loss, they are able to attain much higher throughput. In particular, OvS-DPDK performs 2x better on TCP send and 1.5x better on receive than the in-kernel switch. Moreover, comparing the results of OvS-native and OvS-DPDK reals that the choice of forwarding engine has considerable impact over *packet receiving* performance, where OvS-DPDK increases the throughput by about 30% from OvS-native.

The power consumption results are depicted in Figure 6. The baseline power performance of our NFV testbed is about 6.22 watts when running the host OS alone; running a virtual machine in it will add about extra 2 watts. In terms of the standby power, the in-kernel bridge adds a negligible amount of energy and the OvS-native about 1 watt. The staggering number appears in the case of OvS-DPDK. Even when no traffic is in transit, OvS-DPDK consumes about 3 times more energy than the other two dataplanes, adding 250% energy overhead (15.57 W) on top of the host OS.

When they are used to forward traffic, the discrepancy between OvS-DPDK and the other two dataplanes shrinks, despite the fact that OvS-DPDK still draws about 25% more energy than the in-kernel bridge (400% of the idle host OS). From the core-only power readings (the total accounts for other subsystems such as memory controller, last-level cache, and graphic processor) we found that the CPU cores are the major contributor to the energy discrepancy. OvS-DPDK brings in about 5 watts more consumption in its use of the core. In fact, without using DPDK, OvS-based dataplane achieves similar power consumption at standby and even less when running as compared to the in-kernel bridge.

3.2 Forwarding Engine is the Energy Hog

Multimedia applications are often throughput demanding; using the advanced forwarding engines such as DPDK can benefit the application performance. We confirmed this performance speed-up in the experiments, yet we also showed that high-speed forwarding engine leads to an undue amount of energy consumption. To better understand this issue, we conducted a profiling analysis of OvS-DPDK using `Intel VTune Amplifier` (https://software.intel.com/en-us/intel-vtune-amplifier-xe), a binary instrumentation toolset. Thanks to the open-source nature of DPDK, we are able to conduct detailed, code-level analysis.

To put the dataplane under full load, we launched `hping3` (hping3: https://linux.die.net/man/8/hping3) on both the guest VM and the traffic generator to inject bidirectional traffic at the maximum link speed. First, we found that there is a pmd_thread_main accounts for around 99.7% CPU occupancy (i.e., a full core utilization). By cross referencing to the source code, we found the hot-spots of this thread reside in the dpif−netdev.c:

```
2,850: error = netdev_rxq_recv(rxq, &batch);
```

This function call accounts for 51.6% of the CPU usage. There are also other hot-spots in the same function which we will omit here. By tracing back to the function's caller in pmd_thread_main(void *f_):

```
3,113 for (;;) {
3,114   for (i = 0; i < poll_cnt; i++) {
3,115     dp_netdev_process_rxq_port(pmd,
            list[i].port, poll_list[i].rx);
3,116   }
3,135 }
```

We found that the thread keeps spinning on this code block, indicating it is continuously monitoring and executing the receiving data path. When we repeat the experiment with OvS-native, we did not find such high CPU utilization and thread spinning. We then collected other statistics including the CPI value (*cycles per instructions*) and branch mis-prediction rate for both OvS-native and OvS-DPDK. We found that OvS-DPDK spends more CPU cycles

in fetching and decoding the instructions than actually doing computations (CPI value: 0.693). And the majority of instructions are a result of bad speculation with a lot of mis-predicted branches (i.e., the "if" statements). On the other hand, OvS-native gives lower mis-prediction rate and yields much higher CPI value (1.364). The results demonstrate that OvS-native, as compared to OvS-DPDK, uses CPU more efficiently by spending cycles on useful computations, although on average it spends more time to complete an instruction (due to the use of interrupts).

As a conclusion, we confirm that OvS-DPDK improves its performance by running a spinning thread that keeps polling for any newly received packets in the device buffer to attain better performance. This approach yields higher performance than the traditional, CPU-efficient interrupt based approach. On the other hand, the polling model also leads to an inefficient use of CPU cycles that causes the significantly higher power consumption than the interrupt based approach.

4 How to Manage Power for NFV-based Multimedia Content Distribution?

Our single-host experiments reveal energy inefficiency issues that lie in current NFV dataplane implementations. In reality, dataplanes are often deployed in a multi-host, multi-hop fashion, e.g., multimedia distribution (Figure 1), where energy inefficiency is likely to be amplified or even propagated, making it a network-wide problem. We summarize the following challenges in designing power management frameworks for NFV-based multimedia content delivery:

- At the single-host level, the framework should exploit the commonly available power tuning interfaces on the host machines. Since dataplanes and VNFs are real-time, data-intensive applications, the power tuning must be done carefully without risking performance.
- At the multi-host level, the framework should preserve the Quality of Service (QoS) of multimedia applications, which demands a global view of all transit VNFs performance when applying power tuning.

In this section, we provide the outline of a performance-aware power management framework. We then examined the energy and performance impact of CPU frequency scaling as the power tuning method.

4.1 Overview of a Power Management Framework

In Figure 7, we depict the outline of a proposed power management framework. The framework consists of an agent program deployed on each host server and a centralized power manager. The power manager can be viewed as a control plane service. It should query the NFV manager for the most updated VNF and the host server information. The manager should take charge of generating a *global policy* for network-wide power saving, based on the performance requirements of multimedia applications, stream characteristics, and the current NF status.

4.2 Energy Savings Through CPU Frequency Scaling

Given multimedia traffic are continuous and multimedia middleboxes usually shared by multiple streaming sessions,

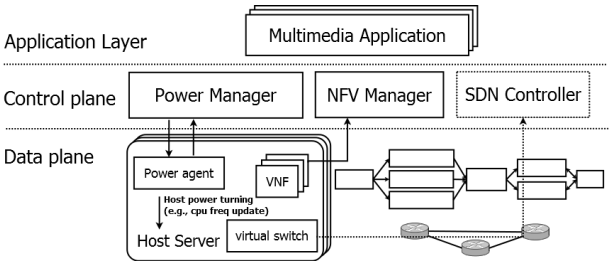


Fig. 7: The proposed power management framework. It complies with the control and data plane separation of SDN. The power agent is deployed on each VNF host server at the data plane and a centralized manager that makes global power-performance tuning decision based on application-supplied policies. Despite sitting at the control plane, the power manager may as well leverage the APIs of other controllers.

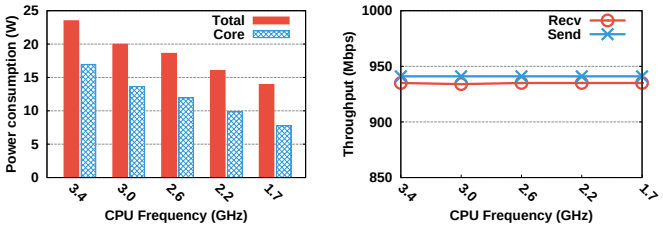
it can be rare to spot an idle time for the middlebox and its underlying dataplane. As such, power tuning methods that rely on OS sleep modes may not function well since they squeeze energy savings from the application idle time. To incorporate the QoS requirements, power tuning methods should also enable fine-grained power-performance trade-off and avoid QoS violations.

We decided that CPU frequency scaling could be a promising power tuning method for NFV host server energy saving. First, the interface has long been supported by modern CPUs and OSes, including most of the server-class machines. Second, it allows frequency tuning at per-core granularity and permits access in the userspace [7] [8]. Although there exist some OS-level power management tools that exploit this interface (e.g., Linux power governors), they cannot be directly used in our case because computation-intensive applications such as DPDK's polling driver may run on a spinning thread and the OS will not be able to distinguish whether it is under heavy load or not.

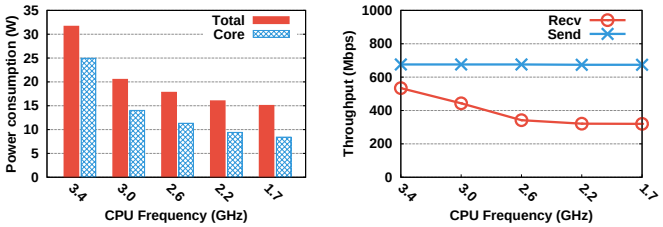
We repeated the iperf experiments in Section 3 with CPU frequency scaling. We devised a script to provide simple frequency control. As a sample power agent, the script probes the target frequency setting by gradually reducing the frequency until it sees the target throughput drops for more than a given threshold, i.e., finding the *Pareto frontier* of the power consumption and throughput. We leave the detailed design and implementation of the full-fledge power agent and manager in our future work.

To illustrate the effect of frequency scaling, we configured the script to iterate through the frequency range and collect the intermediate results. As shown in Figure 8, at the default 1500 MTU, we see a power reduction from 23 watts at 3.4 GHz to under 20 watts at 2.6 GHz and 13 watts at 1.7 GHz with over 43% energy saving. As for the performance, the throughput for both sending and receiving stay the highest across the frequency range. When we change the MTU to 200, the power saving becomes more prominent with the total power consumption halved at 2.6 GHz. On the other hand, the frequency tuning starts to have an impact on the performance on the packet receiving. The throughput drops from around 534 Mbps at the maximum frequency to 320 Mbps at the lowest. We conjecture the performance penalty is caused by the overloaded PMD thread on the packet receiving path. The sending performance remains

unaffected. During the experiment, the frequency control loop consumed less than 1% of CPU and negligible power consumption.



(a) MTU 1500



(b) MTU 200

Fig. 8: Impact of CPU frequency scaling over power consumption and application performance at low packet rate (top, MTU 1500) and high packet rate (bottom, MTU 200).

Based on these preliminary results, we conclude that when the network load on dataplane is moderate or send-bound, it is performance-friendly to lower the CPU frequency to conserve energy; when network load becomes high and receive-bound, the frequency should be raised accordingly to avoid performance penalty. This leads to the idea of *energy proportionality*, the desired property stating that the power consumed in computer systems should be in proportion to how much useful work they complete [8]. To incorporate this property in our framework, the power manager needs to extract the multimedia stream information and forward it to the power agents. A power agent will then exploit this information to estimate the workload in the next time frame and adjust the CPU frequency accordingly.

5 More Discussion on the Performance, Flexibility, and Energy Efficiency Trade-off

Traditional middleboxes rely on hardware specialization, e.g., Application-Specific Integrated Circuit (ASIC), to attain high performance. At the computer architecture level, ASICs differ from general purpose processors in their use of *dataflow* model. It minimizes the overhead of instruction loading, where the program execution solely depends on the availability of input data at the logic gates. Dataflow model is thus well-suited for real-time, data-intensive applications such as packet forwarding and signal processing. On the other hand, hardware specialization often imposes a lacking of *flexibility*. It has longer time-to-market, higher (one-off) provisioning costs, and inconvenient patches and updates, whereas its virtualized counterpart, run on general purpose processors, excel in addressing those issues.

A main focus of the current NFV research is how to make reasonable trade-offs between performance and flexibility [3]. Based on our measurement study, we found *energy efficiency* an equally important dimension in the design space of NFV, particularly when NFV is integrated into Internet-scale services such as multimedia content delivery. Our experiments have shown that performance optimizations in the dataplane could result in substantial energy costs. NFV vendors and customers need to consider these three dimensions together transitioning to NFV.

6 Related Work

Multimedia content delivery has attracted many research interests in recent years. Studies have shown that multimedia broadcasting providers such as Twitch.tv place content processing and distribution servers along the traffic flow path [5], and a majority of them can be in fact categorized as middleboxes [6]. The rise of NFV will make it easier to deploy and manage these middleboxes. It allows network functions to be provisioned as part of the consolidated infrastructure in modern datacenter [2] [4]. For instance, E2 [4] is one of the first frameworks that provide common NFV-related mechanisms for creating and managing network functions in compute clusters.

Performance and flexibility remain the central topic of current NFV research. For example, NetVM [3] allows network functions to be implemented on commodity servers while preserving their line-rate performance. It employs several software-level optimizations such as zero-copy data transmission between VMs to achieve efficient packet processing. Energy efficiency has not drawn full attention in NFV research so far, particularly in the context of multimedia distribution. Among the existing works, Prekas *et al.* [8] designed OS-level mechanisms to achieve energy proportionality for latency-sensitive, data-processing workloads. Song *et al.* [7] proposed a CPU frequency scaling based scheme to conserve energy for video transcoding workloads in the non-NFV context. This article takes the first step towards addressing the energy efficiency in NFV-based multimedia content distribution.

7 Conclusion and Future Work

We investigated the energy cost of advanced NFV platforms for multimedia content delivery. The upsurge of energy consumption is due to the characteristics of multimedia traffic, expensive computations of multimedia network functions and the dataplane energy inefficiency. We proposed a power management framework that leverages CPU frequency scaling to achieve energy saving and meet end-to-end QoS requirements of multimedia applications. For future work, we are going to implement a prototype of the framework and explore the integration with other control plane services including SDN and NFV controllers.

We believe our findings apply to other major NFV platforms. The software components of our testbed are the commonly used building blocks for NFV platforms (e.g., DPDK, KVM), and we have identified the energy issues arise from the software stack as opposed to the specific hardware configuration. Our measurement tools and methodology can be reused on other hardware setups for obtaining quantitative results.

To set up the testbed and reproduce the experiments covered in this article, visit our code repository: `https://github.com/sfu-nml/nfv-mm-testbed`.

REFERENCES

[1] White paper: Cisco vni forecast and methodology, 2015-2020. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2012, pp. 323–336.

[3] J. Hwang, K. Ramakrishnan, and T. Wood, "Netvm: high performance and flexible networking using virtualization on commodity platforms," in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2014, pp. 445–458.

[4] S. Palkar *et al.*, "E2: a framework for nfv applications," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2015, pp. 121–136.

[5] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: A twitch.tv-based measurement study," in *Proc. ACM NOSSDAV*, 2015, pp. 55–60.

[6] B. Carpenter and S. Brim, "Rfc 3234-middleboxes: Taxonomy and issues," *Network Working Group. Ietf*, 2002.

[7] M. Song, Y. Lee, and J. Park, "Scheduling a video transcoding server to save energy," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 2s, p. 45, 2015.

[8] G. Prekas, M. Primorac, A. Belay, C. Kozyrakis, and E. Bugnion, "Energy proportionality and workload consolidation for latency-critical applications," in *Proc. ACM Symposium on Cloud Computing (SoCC)*, 2015, pp. 342–355.

Silvery Fu Silvery Fu is a master student (2015-) in Computing Science at Simon Fraser University, under the supervision of Prof. Jiangchuan Liu. In 2016, he received the B.Sc. degree (First Class with Distinction) from SFU and B.Eng. (Dual Degree Program) from Zhejiang University, both in Computer Science. His contact info: dif@sfu.ca

Jiangchuan Liu Jiangchuan Liu (S'01-M'03-SM'08-F'17) is a University Professor in Computing Science at Simon Fraser University, and an EMC-Endowed Visiting Chair Professor of Tsinghua University. He is an IEEE Fellow and an NSERC Steacie Fellow. He received the inaugural Test of Time Paper Award of IEEE INFOCOM (2015). His contact info: jcliu@sfu.ca

Wenwu Zhu Wenwu Zhu is a Professor and Deputy Head of Computer Science Department at Tsinghua University. He was Research Manager at Microsoft Research Asia before. He was Chief Scientist and Director at Intel Research China from 2004 to 2008. He worked at Bell Labs during 1996-1999. He is an IEEE Fellow, AAAS Fellow, SPIE Fellow, and ACM Distinguished Scientist. His contact info: wwzhu@tsinghua.edu.cn