# On-Demand Clock Boosting for Secure Remote Work System

Justus von der Beek
*School of Computation, Information and Technology*
*Technical University of Munich*
Munich, Germany
e-mail: beek@in.tum.de

Atsushi Shinoda
*Graduate School of Informatics*
*Nagoya University*
Nagoya, Japan
e-mail: shinoda@net.itc.nagoya-u.ac.jp

Hajime Shimada
*Information Technology Center*
*Nagoya University*
Nagoya, Japan
e-mail: shimada@itc.nagoya-u.ac.jp

Hirokazu Hasegawa
*Center for Strategic Cyber Resilience Research and Development*
*National Institute of Informatics*
Tokyo, Japan
e-mail: hasegawa@nii.ac.jp

***Abstract*—The global data center and networking infrastructure is projected to become the largest energy consumer by 2025, with high energy consumption contributing significantly to the climate due to greenhouse gas emissions. The trend of increased digitization accelerated this further, in particularly by Virtual Private Network and Video Conferencing network traffic, leading to higher CO2 emissions. In this paper, we address this challenge by analyzing and reducing the energy consumption of a secure remote working system. We propose a custom clock boosting mechanism using Dynamic Voltage and Frequency Scaling. Our two implementations, utilizing Extended Berkeley Packet Filter and eXpress Data Path, passively listen for new Transmission Control Protocol connections and adjust Central Processing Unit frequency when new employees connect. During idle periods, the frequency is minimized. Through this approach, we achieve up to 28% reduction in energy consumption during high load scenarios, while maintaining virtually no impact on consumption during idle phases. Additionally, the Quality of Service is improved, validating the effectiveness of our strategy.**

***Keywords-VPN; eBPF; XDP; computer networking; energy efficiency***

## I. INTRODUCTION

Due to an increase in remote working and learning since 2019 [1], [2], Virtual Private Network (VPN) connections have been widely adopted, with VPN traffic experiencing a $>200\%$ increase during 2020 lockdowns [2]. This highlights the importance of secure data transfer for businesses and educational institutions. Shinoda et al. have developed an Access Control List (ACL) management mechanism for existing VPN software, which enhances the security of these VPN environments [3], [4]. The system achieves this by restricting access to critical files or servers for inexperienced users, thereby reducing the potential for hacking incidents.

This increase in digitalization poses both a challenge and an opportunity. On the one hand, the new demand of more internet users must be met with larger and more data centers. This also applies to company networks which were expanded during the pandemic [5]. However, it is no news that large data centers require high amounts of energy to operate [6]. Not only is the energy consumption a huge factor in the operating bill, but it also has a potentially high impact on the climate [7]. Liu et al. have estimated that data centers will be the largest global energy consumer in 2025 [8].

On the other hand, digitalization can lead to a decrease in greenhouse gas emissions [9]. In this paper, we deal with this challenge by analyzing the power consumption of our VPN system with dynamic ACL and attempting to minimize its operational energy consumption. In doing so, we strive to reduce energy costs and the environmental impact. To achieve this, we implement an on-demand Dynamic Voltage and Frequency Scaling (DVFS) controlling scheme for our VPN system and measure the potential energy savings and the impact on Quality of Service (QoS).

In Section II, related work is discussed and the optimized system introduced. Section III designs the clock boosting system and introduces two implementation ideas. The Section IV discusses the two implementation ideas in more detail. In Section V, the system is evaluated, followed by a conclusion and further work in Section VI.

## II. RELATED WORK

Because energy consumption is one of the primary cost factors in data centers, there exists a considerable amount of research focused on reducing it.

Krzywda et al. conducted experiments with DVFS schemes for data centers demonstrating its potential to reduce maximal energy consumption by up to 14% [10]. However, their approach involved setting a fixed frequency for a single experiment, which did not allow for dynamic updates at runtime based on application feedback. Additionally, the percent of energy savings they achieved came at the expense of an average performance reduction, which was at least twice as high.

Kasture et al. developed and implemented Rubik, a statistical performance model that utilizes DVFS to reduce the energy consumption in data centers hosting web search functionality [11]. They adapted the frequency to the lowest possible level while maintaining threshold latency. However, their implementation is not available online to reproduce and evaluate against. Rebuilding this approach is out of scope for this paper. To keep the power consumption low, our idea can operate by keeping a counter of active Transmission Control Protocol (TCP) connections, making the expensive calculations of the paper needless.

The summary paper by Zhu et al. emphasizes the potential of DVFS schemes in conjunction with advancements in AI-driven prediction algorithms [12]. At present, the main disadvantages lie in the costs and computation difficulty of the prediction algorithm. In our work, we aim for a simple feedback solution with minimal calculation cost to reduce the energy consumption impact on the system.

### A. *Secure Remote Work System Overview*

In the following, we provide a brief introduction to the secure remote work system that aim to optimize [3]. Figure 1 offers an overview of the key components for our paper, found in the system. This setup involves simulating a company network constructed by a Software Defined Networking (SDN). When an employee connects to the company network via VPN, the system applies a fine grained ACL to the SDN controller. The ACL's parameters depend on the user's reliability, enabling the blocking of access to high-risk data for unreliable users. The primary objective is to enhance the system's security by additional layers of security.

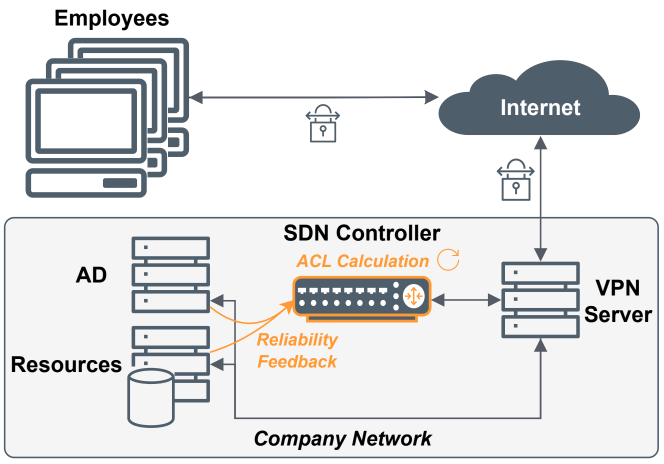

Figure 1. The mock up company network of Shinoda et al. We focus on the machine marked in orange, hosting the SDN-Controller because the CPU bound ACL computations allow for power savings.

To achieve this, a miniature network with VPN server, so-called VPN clients, active directories containing company mock data, and other resources is built. While not all parts of the network are relevant for our analysis, we have omitted them here for the sake of simplicity. If a user is not considered reliable enough, their access to files or servers will be blocked through the ACL permissions.

The main objective of this paper, is to reduce the overall system energy consumption by implementing a dedicated DVFS mechanism. The method also tries to minimize any adverse effects on performance. Given that the calculation of the ACL requires some time and involves CPU-heavy computations, our focus was mainly directed towards exploring potential power savings in the SDN-Controller.

## III. Design of On-Demand Clock Boosting for Secure Remote Work System

Our SDN-Controller only operates at the beginning and end of a VPN connection. One significant challenge for our use-case lies in the uncertainty of when a new client will connect. Acknowledging that we cannot know the future, we opted for a reactive approach. Developing an extensive network prediction algorithm or model was beyond the scope of this work. Especially, as subsequent results revealed that the system achieved satisfactory latency with on-demand boosting. Nevertheless, we expect achieving better result by responding swiftly to any networking event. In Linux terms, our aim was to find a method positioned as low in the network stack as possible. Linux kernel version 3.15 and later features Extended Berkeley Packet Filter (eBPF) programs, which perfectly align with our idea. These programs can be attached at a low level in the network stack and have low overhead, making them an ideal fit for our requirements.

### A. *Extended Berkeley Packet Filter (eBPF)*

Extended Berkeley Packet Filters enable the execution of special sandboxed programs within the kernel space, all without requiring any kernel modifications or modules [13]. These eBPF programs are coded in a C-like syntax and are compiled into what is known as eBPF bytecode. Upon loading, this code undergoes thorough verification and checks for potential errors such as out-of-bounds memory accesses or potential infinite runtime scenarios. The presence of these checks ensures that eBPF programs cannot crash or cause deadlocks, making the execution inherently safe [13]. Furthermore, the eBPF loader provides the flexibility to switch the specific hook to which an eBPF program is attached. For instance, one such hook could be the Linux networking stack, while other attachment points include kernel tracepoints or system calls.

### B. *eXpress Data Path* (XDP)

Due of the flexibility and advantages of eBPF, the eXpress Data Path (XDP) hook point was developed. It enables the attachment of eBPF programs at the driver or hardware level, bypassing most of the network stack. As a result, this approach offers improved bandwidth and higher packet rates compared to the default kernel [14]. Nonetheless, it is important to note that not all devices and drivers support the XDP hook point. To address this limitation, an emulation mode was introduced after the Linux sockets module, allowing eBPF programs to be attached at an XDP hook without offering the advantage in speed.

For easy communication between the program running in the kernel space and user space, eBPF maps are available [13]. These data structures are allocated in shared memory regions, facilitating read and write operations from both locations. Utilizing these maps, both programs can effectively communicate with each other, or the XDP program can maintain and store state information.

It is worth mentioning that we deliberately chose not to use the Data Plane Development Kit (DPDK). While DPDK's busy polling design might lead to even faster reaction times than XDP, it would come at the cost of significantly increased energy consumption. At least one CPU core would constantly operate at 100% utilization [15], rendering any potential energy savings irrelevant.

### C. On-Demand Clock Boosting Design

The basic idea behind the clock boosting service is simple: we aim to provide the best available QoS while consuming as little energy as possible. To achieve this, we focus on utilizing the lowest feasible CPU frequency. In our test environment, the SDN switch is the only entity that connects to the SDN controller. No other connections are made to the machine. Consequently, all incoming TCP connections to the SDN controller originate from the SDN switch and are followed by the ACL calculation. Due to this fact, we can employ a straightforward check for new TCP connections directed to our controller. Upon detecting a new connection, we can boost the CPU frequency, thereby increasing the processing speed of the ACL calculation. Once the connection is reset, indicating that the processing is complete, we can then reduce the frequency back to the energy-saving level. Given that the system servers exclusively as an SDN-Controller, we have the condition to aggressively switch the CPU frequency back to power-saving mode without any performance implications. In the following, we propose two ideas to realize this design. One is based on the BPF Compiler Collection (BCC) [16] library while the other idea uses XDP.

### D. Idea 1: BCC and cpupower

In our first approach, we have devised a system that attaches itself to the TCP/User Datagram Protocol (UDP) hook in the network stack, as illustrated at label 2 (Trigger on packet) in Figure 2. The process involves a new packet arriving on the network interface (label 1 in Figure 2), passing through the Linux network stack, and reaching the TCP/UDP module (label 2). Each time a packet passes through this module, it triggers the `tcpaccept` program which passively listens here. The `tcpaccept` program then invokes a shell script located in user space at label 3 in Figure 2. The shell script, temporarily increases the system's frequency for a short duration of 0.7 seconds to accelerate the calculations. We found this duration to be sufficiently long enough to maintain an equal QoS level. After this duration elapses, the performance governor is reverted to powersaving mode and the CPU frequency set to the minimal achievable frequency.

### E. Idea 2: XDP, eBPF Maps and C-based Frequency Switch

The second approach utilizes XDP and an architecture overview is presented in Figure 3. Instead of employing

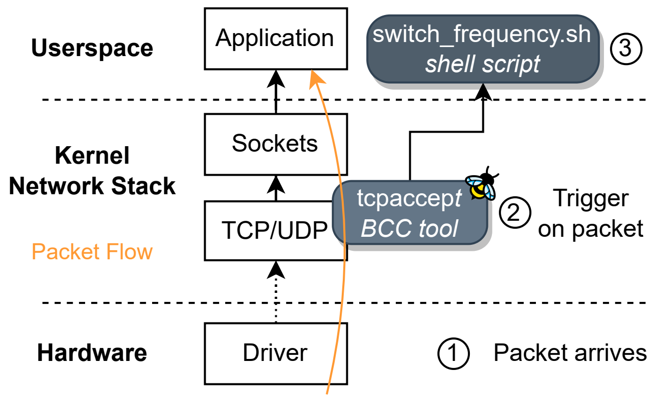

Figure 2. The eBPF BCC-based solution involves the `tcpaccept` program passively listening for incoming connections. When a connection is detected, it triggers a shell script to increase the frequency. After a set duration, the frequency is lowered again.

`tcpaccept` at the TCP/UDP module, we attach an eBPF program at the XDP hook, as shown in Figure 3 at label 1.

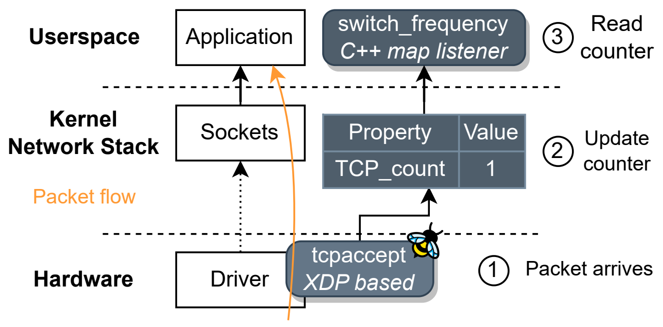

Figure 3. The XDP design involves an eBPF program that passively listens for new TCP connections and communicates with the user space frequency switch through shared memory maps. This communication also enables the system to detect disconnects.

Every time a packet arrives, the eBPF program is triggered and updates a counter variable in a shared memory map (labeled 2 in Figure 3). At the user space level, marked with label 3, we read the counter value and increase the frequency when at least one client is connected. Conversely, if all clients are disconnected, the frequency is decreased to the powersaving level.

## IV. IMPLEMENTATION

In the following section, we will discuss the implementation details of the two solutions in greater detail and highlight advantages and disadvantages of each.

For our first idea we built upon Brendan Gregg's `tcp-accept` program [16]. It was originally designed to print statistics each time a new TCP connection is made. Since the executed user space program can be easily modified, we decided to base our implementation on this tool. For each new connection, we execute a shell script that utilizes `cpupower frequency-set` to switch to the highest available frequency and change the power governor to the `performance`

```
1: if $packet$ is TCP then
2:     if $packet.flags$ is SYN then
3:         $map[TCP\_count] \leftarrow map[TCP\_count] + 1$
4:     else if $packet.flags$ is FIN then
5:         $map[TCP\_count] \leftarrow map[TCP\_count] - 1$
6:     end if
7: end if
```

Figure 4. The XDP TCP packet listener program

```
1: $prv\_count \leftarrow 0$
2: while $true$ do
3:     $count \leftarrow map[TCP\_count]$
4:     if $count \geq 1$ and $prv\_count = 0$ then
5:         $boost\_frequency()$
6:         $prv\_count \leftarrow count$
7:     end if
8:     if $count = 0$ and $prv\_count \geq 1$ then
9:         $reset\_frequency()$
10:        $prv\_count \leftarrow count$
11:    end if
12: end while
```

Figure 5. The XDP program user space map listener

mode [17]. After a 0.7-second interval, the frequency is reverted to the lowest possible value and the governor is reset to `powersave` mode. To allow for changes to the frequency, this script must be run as root. In total, the implementation cost of this approach is less than 40 lines of code.

In contrast, the second solution utilizes XDP to parse incoming Ethernet packets and identify those carrying the TCP protocol. If a TCP packet is found, it is further checked for the SYN or FIN flags. When a SYN flag is detected, the number of active TCP connection is increased, while a FIN flag results in a decrease of active connections as shown in Figure 4. We maintain a counter in the shared memory map between XDP and user space which the XDP program updates (lines 3 and 5 in Figure 4). This counter informs the user space program about the current active connections and enables it to make decisions about the frequency to use. If there is one or more TCP connection, we switch to the performance governor and the highest frequency as shown by the if at line 4 in Figure 5. However, if there are none, we reset the governor to powersave mode and switch back to the lowest frequency (line 8 onward in Figure 5). Both of these actions are accomplished by writing the frequency and governor directly into the *sysfs*. This solution requires root privileges to load the XDP program and roughly 600 lines of new implementation in total. The user space listener can be loaded without root privileges required.

### A. Disadvantages of Proposed Systems

One disadvantage of the first system is that in only reacts to TCP connections and does not monitor disconnects. Additionally, the solution introduces an extra of indirection through the shell script instead of directly writing to the MSR registers. This indirection adds time from the connection detection to the actual frequency increase. On top, the user space handling of the BCC tool is written in python which is an interpreted language with a higher overhead than C. It requires more resources than a C program and has a slower reaction time.

The major drawback of the XDP-based system, in our case, lies in the lack of XDP driver support from our Ethernet card. As a result, we have to rely on the emulation mode after the Socket module in the Linux network stack. Unfortunately, this leads to a reduction in reaction time instead of the desired improvement.

### B. Advantages of Proposed Systems

Both systems share the advantage of low impact during idling since the programs are only triggered when an event occurs and they do not execute otherwise. This idle-aware design reduces the energy consumption making it an efficient solution.

Additionally, the XDP solution provides the advantage that it listens to disconnect events. This capability enables us to determine the duration for which we need to increase the CPU frequency. When multiple connections are active, we increase the frequency further to help handling all clients. Given that connections are only established when SDN-Controller decisions need to be made, the higher frequency will accelerate the processing enabling us to maintain a better QoS.

## V. Evaluation

This section is split into the evaluation setup and the testing results.

### A. Evaluation Setup

The SDN-Controller system runs on a HP ENVY x360 laptop with AMD Ryzen 3700U CPU with 4 cores, SMT enabled, TDP of 15 W and 16 GB RAM. We tested both with Turbo Core enabled and disabled to see the impact on the power consumption and performance.

The driver (`acpi-cpufreq`) allows for 3 frequencies with 1.4 GHz, 1.7 GHz and 2.4 GHz. Furthermore, the `performance`, `powersave` and `ondemand` governors are available [18]. We are using RockyLinux running kernel version 6.3.9-1. The ryu-manager version is 4.34. The Ethernet card is a Buffalo LUA4-U3-AGTE-NBK USB3.1 Gigabit Ethernet Card with driver version ax88179_178a. The VPN-Server deploying the VPN functionality is a NEC IX2310 running firmware version 10.6.63. Energy Consumption is measured via the Running Average Power Limit (RAPL) interface using the `turbostat` tool. Different research has shown that this interface is accurate enough for comparison of benchmark runs on the same system [19]. We are using 9 clients that simulate employees wanting to connect to the company internal network via VPN. All clients are using Windows 10.

The benchmark consists of all clients running a powershell script that uses `rasclient` to connect and disconnect to the company network as fast as possible. The time measurement

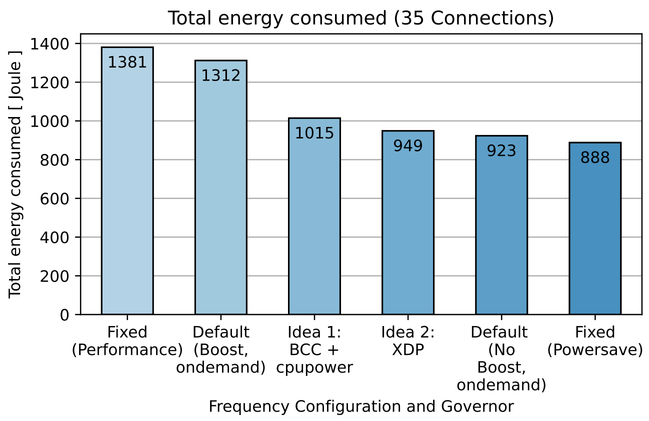

Figure 6. The VPN connection test total energy consumption. Both ideas can reduce the consumption by more than 23% compared to the default case with ondemand governor and Turbo Core enabled (named Boost in the Figure).

is done via the powershell internal `Measure-Command`. With the VPN connection and disconnection time, we try to measure a relevant QoS metric because users will notice a delayed VPN connection time as soon as they start working. Furthermore, this test should show the impact on the performance due to modified CPU frequency handling. File transfers, however, are not impacted by the VPN-controller and are therefore not tested.

Currently, the Ethernet network card of our SDN-Controller does not support XDP driver offload. Therefore, the slower SKB attachment mode was used during development and evaluation. We would expect better results for the native XDP mode because of a faster reaction time.

### B. Evaluation Results

The accumulated power consumption of our benchmark runs for different frequency configurations is plotted in Figure 6. Both the BCC and XDP version of our proposed system can reduce the energy consumption during the high load scenario by more than 23% compared to the `ondemand` governor with Turbo Core enabled. We consider this the default version because this configuration is loaded on start up without any modifications to the system. The XDP version is less resource demanding than the python implementation, decreasing the consumption further by 5%. We make two noticeable observations. First, the performance mode only increases the power consumption by 5% hinting a very aggressive CPU frequency selection by the Operating System (OS). Secondly, disabling the frequency boosting technology saves 2% more energy than our XDP implementation. Krzywda et al. [10] approach would likely achieve results similar to the `ondemand` governor with Turbo Core disabled in Figure 6.

When looking at the idle consumption in Table I the proposed system does not increase the total consumption. Due to the event based triggers of eBPF and XDP the impact on energy consumption is only during high load phases. There are no energy savings during the idle periods but also no additional costs. This is important because most of the time the system will stay in an idle phase. An increase in the consumption during this phase would therefore outweigh savings during high load.

TABLE I
THE IDLE POWER CONSUMPTION (120S IDLE)

| Mode | Default | XDP | BCC |
|---|---|---|---|
| **Energy Consumption** | 256.62 J | 262.02 J | 266.62 J |
| **Relative Consumption** | 100% | 102% | 104% |

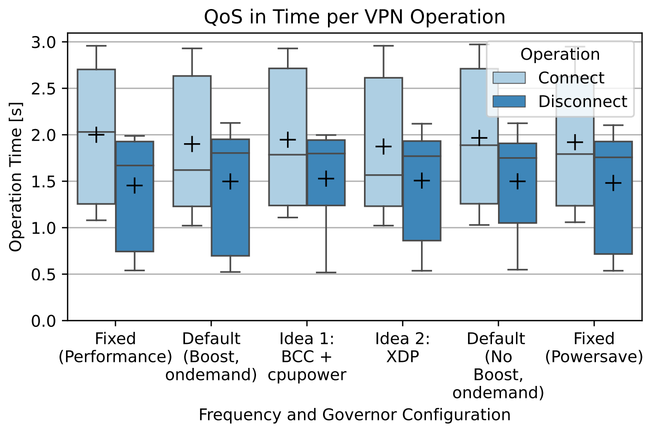

Figure 7. The VPN-Connection test connection time. Outliers larger than 4s and smaller than 0.5s were excluded. The + denotes mean, the horizontal bar median. The whiskers show the 95th percentile.

Switching the CPU frequency has an impact on the time it takes to calculate the ACL rules and inform the SDN-Switch about the final decision, which determines whether access is allowed or blocked. Figure 7 presents the mean connection and disconnection times in seconds, accumulated from all 9 clients, in a boxplot. The x-axis displays the different configuration modes, while outliers greater than 4 s or smaller than 0.5 s are excluded from the analysis as they indicate errors. The mean value is indicated by the + marker, the horizontal line indicates the median value. Interestingly, the results show that there are no significant differences in connection and disconnection times among the various configuration modes. However, our first implementation idea increases the connection times by 2.4% while achieve less power savings than the `powersave` and no boost configuration, as evidenced in Figure 7. For the default powersaving mechanisms a 1-3% increase in mean connection time can be observed. For the XDP version, the mean connection time is 1.87 s compared to 1.90 s in the default no boost case and 1.97 s in the powersaving configuration. The difference in median and mean value for the XDP configuration seem to come from slower outliers which degrade the median connection time of 1.55 s. Effectively, we spend 5% more energy for a 3% mean connection time decrease compared to the no boost configuration. For the XDP implementation we would expect better results when switching to the driver offloaded eBPF program. This could decrease the reaction time and lead to faster connection times.

The chosen benchmark marks a best case scenario for our proposed system. Because the CPU frequency in idle phases is automatically reduced by the operating system itself we improve due to the more aggressive frequency switching. Furthermore, we benefit from the knowledge, that the CPU is not required in the intervals between connections allowing for the fast frequency reduction. Because the system is solely used as a SDN-Controller the power saving is not impacting performance of any other program.

## VI. Conclusion and Future Work

In this paper, we presented two implementations aimed at reducing the energy consumption of a secure remote work system. The BCC-based design, along with the cpupower tool, achieved a decrease in energy consumption compared to the default power governor, but better results were obtained by disabling Turbo Core. The XDP version, due to its lower resource impact in user space, further reduced power consumption. Compared to the default configuration, a 1.5% decrease in connection time was achieved, and compared to the disabled Turbo Core configuration, connection times were 5% faster. However, this required approximately 3% more energy compared to the ondemand governor with disabled Turbo Core.

Future work in the system could involve migrating the SDN-Controller into a virtualized environment to reduce required components and idle power consumption, while still meeting QoS requirements under high workload scenarios. Additionally, we expect the frequency switch to work on the SDN-Switch, further reducing consumption during file transfers by throttling the frequency to match maximum network bandwidth. Another potential method of power reduction would be to adapt devices to workday patterns and implement high power-saving sleep modes for infrequently used computers.

## Acknowledgment

This research was supported by JSPS KAKENHI Grant Numbers JP23H03396, JP19K20268.

## References

[1] T. Favale, F. Soro, M. Trevisan, I. Drago, and M. Mellia, “Campus traffic and e-learning during COVID-19 pandemic,” vol. 176, article 107290.

[2] A. Feldmann *et al.*, “The lockdown effect: Implications of the COVID-19 pandemic on internet traffic,” in *Proceedings of the ACM Internet Measurement Conference*. ACM, pp. 1–18.

[3] A. Shinoda, H. Hasegawa, Y. Yamaguchi, H. Shimada, and H. Takakura, “Implementation of access control method for telecommuting communication based on users’ reliability,” in *Proceedings of Computer Security Symposium*, pp. 840–847.

[4] H. Hasegawa and H. Takakura, “A dynamic access control system based on situations of users,” in *Proceedings of the 7th International Conference on Information Systems Security and Privacy*, pp. 653–660.

[5] F. Almeida, J. Duarte Santos, and J. Augusto Monteiro, “The challenges and opportunities in the digitalization of companies in a post-COVID-19 world,” vol. 48, no. 3, pp. 97–103.

[6] Z. Song, X. Zhang, and C. Eriksson, “Data center energy and cost saving evaluation,” vol. 75, pp. 1255–1260.

[7] J. Ferreira, G. Callou, A. Josua, D. Tutsch, and P. Maciel, “An artificial neural network approach to forecast the environmental impact of data centers,” vol. 10, no. 3, p. 113.

[8] Y. Liu *et al.*, “Energy consumption and emission mitigation prediction based on data center traffic and PUE for global data centers,” vol. 3, no. 3, pp. 272–282.

[9] J. Sutton-Parker, “Determining commuting greenhouse gas emissions abatement achieved by information technology enabled remote working,” vol. 191, pp. 296–303.

[10] J. Krzywda, A. Ali-Eldin, T. E. Carlson, P.-O. Östberg, and E. Elmroth, “Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling,” vol. 81, pp. 114–128.

[11] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, “Rubik: fast analytical power management for latency-critical systems.”

[12] H. Zhu *et al.*, “Future data center energy-conservation and emission-reduction technologies in the context of smart and low-carbon city construction,” vol. 89, p. 104322.

[13] What is eBPF? an introduction and deep dive into the eBPF technology. [Online]. Available: https://ebpf.io/what-is-ebpf/

[14] D. Scholz *et al.*, “Performance implications of packet filtering with linux eBPF,” in *2018 30th International Teletraffic Congress (ITC 30)*. IEEE, pp. 209–217.

[15] S. Gallenmuller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, “Comparison of frameworks for high-performance packet IO,” in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, pp. 29–38.

[16] B. Gregg, S. Goldshtein, T. Qin, and H. Chen, “BPF compiler collection (BCC).” [Online]. Available: https://github.com/iovisor/bcc

[17] Linux User’s Manual, *cpupower-frequency-set(1)*.

[18] D. Brodowski. CPU frequency and voltage scaling code in the linux(TM) kernel. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[19] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, “RAPL in action: Experiences in using RAPL for power measurements,” vol. 3, no. 2, pp. 9:1–9:26.