

## A new approach of scalable traffic capture model with Pi cluster

Kristoko Dwi Hartomo<sup>1,3</sup>, April Firman Daru<sup>2</sup>, Hindriyanto Dwi Purnomo<sup>3</sup>

<sup>1</sup>Faculty of Information Technology, Satya Wacana Christian University, Salatiga, Indonesia

<sup>2</sup>Faculty of Information Technology and Communication, Semarang University, Semarang, Indonesia

<sup>3</sup>Faculty of Information Technology, Satya Wacana Christian University, Salatiga, Indonesia

---

### Article Info

#### Article history:

Received Mar 14, 2022

Revised Sep 14, 2022

Accepted Oct 8, 2022

---

#### Keywords:

Cluster computing

Internet of things

Message passing interface

Parallel processing

Scalability

---

### ABSTRACT

The development of the internet of things (IoT), which functions as servers, device monitors, and controllers of several peripherals inside the smart home, eased workload in many sectors. Most devices are accessible through the internet because they communicate with wired or wireless interfaces. However, this feature makes them prone to the risk of being exposed to the public. The exposed devices are an easy target for the third party to launch a flooding attack through the network. This attack overloads the system due to the low processing capability, thereby interrupting any running process and harming the device. Therefore, this study proposed a scalable network capturing model that utilized multiple Raspberry Pi boards in parallel to monitor the network traffics simultaneously. An isolated experiment was used for evaluation by running simultaneous flooding attacks on each device. The result showed that the model consumed 30.44% more memory with 14.66% lower central processing unit (CPU) usage and 3.63% faster execution time. This means that this model is better in terms of performance and effectiveness than the single capture model.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



---

### Corresponding Author:

April Firman Daru

Faculty of Information Technology and Communication, Semarang University

Soekarno-Hatta Street, West Tlogosaro, Semarang 50196, Indonesia

Email: firman@usm.ac.id

---

## 1. INTRODUCTION

The invention of the internet of things (IoT) has reduced users' workload in various sectors through the embodiment of simple processing and communication capability. Devices connected to the internet can be monitored and controlled easily by their users, anywhere and anytime. These devices are connected and communicate with a wired or wireless network and are equipped with better processing capability to handle many scenarios. These devices are called single board computers (SBC), with better processing time than the microcontroller-based device. In addition, this device has an operating system installed which eases the operation and management. IoT, which increases productivity, is suitable for many situations such as homes, offices, and industries [1]. This technology is applicable in many situations such as inventory management, production, logistic, retail, and resulting many industries utilize it in their system [2]. However, the available devices are vulnerable to outside attacks [3], with most intrusions launched in specific protocols, such as transmission control protocol (TCP) and internet control message protocol (ICMP). These protocols have layer-specific network attacks that require a detection system [4].

SBC can be equipped with the intrusion detection system (IDS) to detect any intrusion targeting the IoT devices with either signature [5] or behavior-based detections [6]. This device monitors any information that flows within the network and analyzes it for intrusion possibilities. However, its difficulty lies in the typical capture method, as most intrusion detection systems such as Snort are limited in monitoring

capability. The system is only limited to a single network interface to detect intrusions with snort used to run several processes to monitor multiple network interfaces have lower efficiency. Figure 1 is the current traffic capture model.

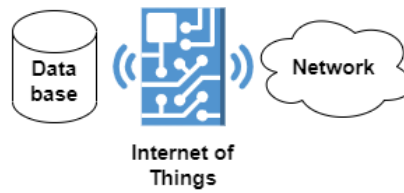


Figure 1. The current traffic capture model

The most common traffic capture model consists of one network interface managed by the system found within a network and stored in the data set [7]. Several studies focused on efficient packet capture and found that the model benchmark is capable of storing packets with 120 Gb/s speed [8]. High-speed network packet capture has also been verified to understand the efficiency. Linux packet capture’s traditional approach is limited only to the rate of packet capture, resource usage, and principles [9]. Some studies also proposed using Moloch-based packet capture to preserve data for forensics. However, this study found that the method is simpler and less complicated than typical network capture, which poses a problem when big data is included [7]. It focused on multi-channel multi-radio attack detection with a proposed algorithm called the cumulative sum algorithm to monitor every network change. A single interface was used to capture traffic within the network by utilizing the WinPcap library also utilized a single interface instead of a multi-interface [10]. Although the proposed method can capture traffic, it has some defects, such as the limitation of a specific operating system [11].

Security is a serious matter in IoT, specifically when the device is open to the public. One of the risks is service disruption caused by denial of service (DoS) attacks [12]. The signature-based intrusion detection system is one of many proposed systems that use a list of intrusion characteristics. Several studies made wireless sensor networks detect intrusion wirelessly by implementing IDS and reporting mechanism through message queuing telemetry transport (MQTT) protocol in the Raspberry Pi device. The previous model has capabilities to sense, predict and prevent DoS attacks in the network. Meanwhile, the other model added encryption to the IDS database for better security by using a single communication channel to detect intrusions [13], [14].

Most studies only utilized a single interface inside the device to capture traffic within the network. However, this type of model has some weaknesses, such as being capable of capturing one wireless network only, with the addition of one or more devices to solve the problem. This increases the workload to compile the data, with device controls managed separately by the user, without using efficient hardware resources for the same workload. Figure 2 explains how adding extra devices or processes can increase the workload for data compilation and further data analysis [7].

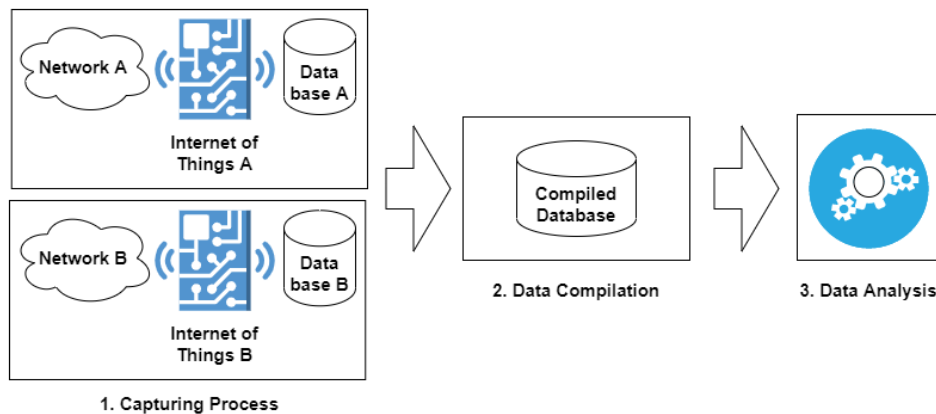


Figure 2. Data compilation process in the current model

The current model monitoring capability is limited to one interface, with the occurrence of network intrusion in any existing network. In this situation, the current model cannot detect intrusions in the networks. Unmonitored connections become an easy target for the third party to attack and destroy data inside the company system. Besides, the inefficiency of the current model may cause several problems, such as the rise in the cost of network security and the workload for analysis.

Due to these reasons, this study proposed a new approach as the novelty by improving the current model by adding simultaneous network traffic capture with two or more Raspberry Pi computers. Unlike the previous models that only utilized one hardware, the proposed model relied on a cluster computing concept where many Raspberry Pi communicate with each other to solve a complex task known as Pi cluster. The proposed model can monitor multiple networks with a scalable feature and add more hardware to increase the capture capacity.

## 2. THE PROPOSED METHOD

This section explains the proposed method used in the study, which consists of several parts, such as the network topology, the proposed algorithm of the model, and the evaluation method. The network topology explains the inter-device connectivity, such as the type of device, its role, and the communication protocol used in the network. The proposed algorithm explains how the model captures multiple traffics in the network while the evaluation method evaluates its performance.

### 2.1. Hardware specification

The proposed models were determined using one Raspberry Pi Zero W and three Raspberry Pi 3B plus. The Raspberry Pi Zero W specification is a 1 GHz single-core CPU, 512MB of RAM, wireless network, and ethernet HAT module through GPIO and micro USB data port. This computer act as a master device in the proposed model, while the Raspberry 3B plus is a 1.4 GHz quad cores CPU, 1 GB of RAM, dual-band wireless network, ethernet, and USB modules. Besides, this study builds one isolated wired network and an open wireless network. All hardware has 32GB of micro USB installed as storage. The official Raspberry Pi OS was also installed into the operating system of a desktop containing Python 3, with additional libraries such as pyshark (network capture), psutil (performance benchmark), mpi4py (message passing interface wrapper), and OpenMPI.

### 2.2. Network topology

This study proposes a simultaneous network traffic capture model with multiple IoT devices. This model used many devices compared to previous models that used a device to capture traffic networks. With this configuration, it is possible to capture traffic from several network connections with a single instruction [15]. To make this model work, it must have a single master and several worker devices, which handles the process control and perform capture process, respectively.

Figure 3 explains the topology model of multi-interface and multi-device traffic captured where master IoT managed three worker devices. In this study, the master device is assigned as a process manager to control the running processes in each Worker device. It sends a query to store the captured data in the database server, which receives data from the capture process. This eliminates the data compilation process with the communication between the master and worker devices using open message passing interface (OpenMPI) through Python [16]–[18]. This interface spreads processes according to the allocated device processors available in the system before the commencement of the capture process [19]. According to preliminary studies [18], [19], the capture effectivity increases by running multiple processes in parallel limited to one wireless network interface. The topology requires properly implementing an algorithm and running inside the device. This algorithm allows interface control for each connected device and compiles captured traffic data into one.

### 2.3. The proposed algorithm

The proposed algorithm copies the concept of single intrusion multiple data (SIMD), where a single instruction can operate multiple devices to process several data. This study tries to achieve this by implementing a message passing interface (MPI) protocol into the models. With this protocol, the master device can instruct numerous devices to capture multiple networks simultaneously [20], [21]. Figure 4 illustrates the SIMD concept with MPI.

Figure 4 explains how the algorithm manages the traffic capture process using one master and many worker devices. The algorithm was unable to differentiate the master from workers. Therefore, this study configures the rank assignment manually through a machine file. It listed the master device as the first, which is ranked zero (0) in the model, while the remaining follow suit ( $rank > 0$ ). This ranking system helps this

study assign a role in the algorithm, which decides what process to run with the master and worker devices ranking 0 and > 0. The process control in the master device monitors contains warning and error information for the users to carry out the termination commands. The master device will monitor MPI Abort flags when triggered by the worker process. When the user aborts the process, the master device sends a termination command to all running processes.

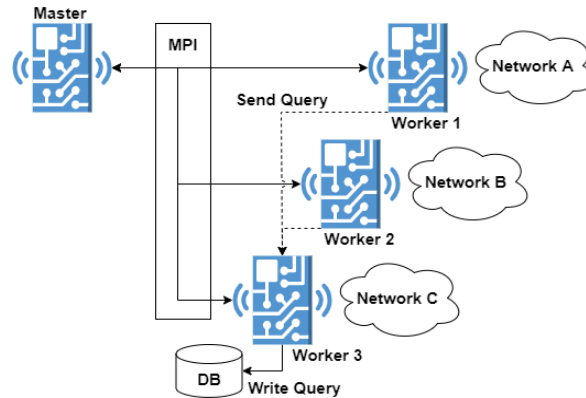


Figure 3. Pi Cluster-based traffic capture model

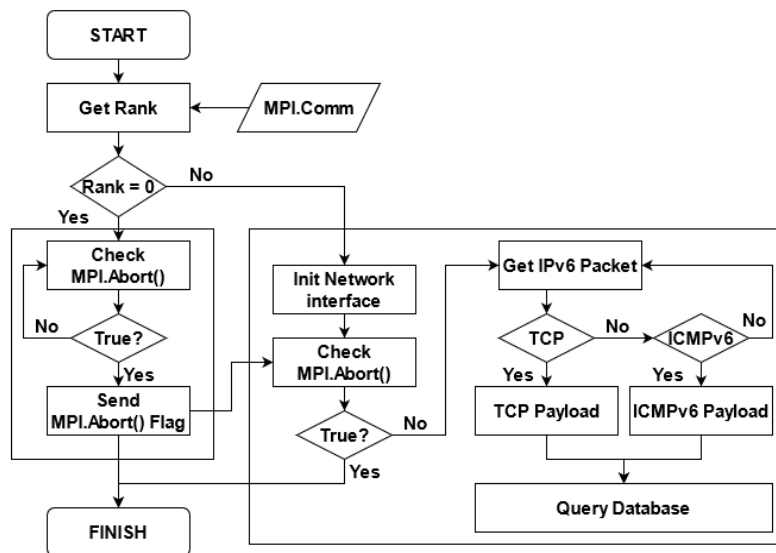


Figure 4. Scalable traffic capture algorithm

In the worker devices counterpart, the process started by initializing the wireless network interface for packet capture. Furthermore, this algorithm runs a loop that monitors the MPI abort flag by fetching an IPv6 packet from the network interface for further data extraction. The process continues to extract the captured data to filter TCP and ICMP protocols. When the captured packet has TCP protocol, the process extracts its parameters, such as the source and destination addresses, protocol, source and destination ports, sequence packet, window size, and payload as TCP characteristics. However, when the captured packet is an ICMP packet, then the algorithm extracts the source and destination addresses, protocol, ICMP type, and data. The result of all processes queried to the database for long-term storage, and at the end of the loop, the algorithm monitors the MPI abort flags from the master device and terminates the process. Meanwhile, the communication exchange between devices must be separated using the Ethernet network interface connected to a local area network to avoid capturing performance degradation and communication interruptions during the process. Compared to the typical traffic capturing model, the proposed model has several advantages like centralized process control, support scalability with more than two devices, simultaneous capture of different wireless networks, and single data set compilation.

## 2.4. Model evaluation and validation

This study evaluates the algorithm's efficiency of IoT by comparing the proposed model with a typical traffic capture model. The evaluation process includes performance benchmarking by configuring experiments to obtain the right data [22]–[24]. In this experiment, the intruder attacks three wireless devices by flooding them with TCP and ICMP packets simultaneously. These devices receive attacks and store them in the database server using the querying method. Every captured process in each model records specific benchmark data stored in its memory called unique set size (USS) in megabytes [25], CPU usage, and execution time. These benchmark indicators are Linux-based performance accessible to the users and suitable as evaluation parameters. The internal performance of IoT boards also plays a role in the capturing process, while the network performance has many external factors. Therefore, the internal board performance is selected as evaluation parameters [26], [27]. The execution time explains how fast the model captures a packet, which also helps to calculate the rate of the model, as shown in (1).

$$\text{Capture Rate} = \frac{\text{Num of Packets}}{\left(\frac{\sum \text{Exec Time}}{1000}\right)} \quad (1)$$

As shown in (1) calculates the capture rate by dividing the total numbers of the captured packet by the sum of execution time in seconds, where 1 millisecond is equal to 1e-3 second. The result of this calculation is useful for determining the capture rate of both models. Before the experiment started, this study made two hypotheses based on benchmark indicators, as shown in Table 1.

Table 1. Benchmark indicator hypothesizes

H	Hypothesis
H0	X Scalable Means >= X Non-scalable Means
H1	X Scalable Means < X Non-scalable Means

The hypothesis above is used to decide whether the scalable model utilized more hardware resources than the non-scalable and vice versa. The X values obtained during the experiment are the benchmark indicators, such as USS, CPU usage, and execution time. The benchmark result alone is insufficient to provide evidence for the evaluation. Therefore, this experiment uses the independent left-tailed test. This study used (2) to obtain the  $t$  value and the critical data according to the student's  $t$ -table. This independent left-tailed test helps the validation process of the proposed model.

$$t = \frac{(\bar{X}_1 - \bar{X}_2)}{\sqrt{\left[\frac{SS_1 + SS_2}{N_1 + N_2 - 2}\right] \left[\frac{1}{N_1} + \frac{1}{N_2}\right]}} \quad (2)$$

The equation of the independent one-tailed test consists of several parts, which calculate the sum of the squared values, data size, mean, and the sum of squares. The first step to obtaining the independent test value denoted as  $t$  is subtracting the scalable model's mean ( $\bar{X}_1$ ) from the non-scalable ( $\bar{X}_2$ ). The  $SS_1 + SS_2$  in the lower part of the equation, refers to the addition between the sum of the square of the scalable ( $SS_1$ ) and the non-scalable ( $SS_2$ ) models. Let  $N_1 + N_2 - 2$  be the degree of freedom between and pooled standard error ( $\frac{1}{N_1} + \frac{1}{N_2}$ ) set to the number of the scalable ( $N_1$ ) and the non-scalable ( $N_2$ ) model data. The  $t$  value is obtained by dividing the subtraction of the mean in the upper equation with the square root of the product in the lower equation. After obtaining the  $t$  value from the equation, the  $t$  critical value is determined using the student's  $t$ -table with an alpha value of 0.05. When the value of  $t$  is outside and within the  $t$  critical value, the hypothesis is accepted and rejected, respectively.

## 3. METHOD

The experiment process has specific configurations to ensure that the model is reproducible with fair evaluation benchmark results. The first step is configuring the topology because, without proper network connectivity, the experiment process will not run as intended. Figures 5(a) and 5(b) are the two different topologies with varying configurations designed in this study.

The evaluation process was carried out using a benchmark with high-performance computing systems [24], [28], [29]. The model's evaluation is an experiment with a total of four Raspberry Pi boards and one computer. The master and workers handle process spawning and capture the traffic while the

database stores data. The computer acts as the attacker on the boards, while the master is assigned to a single-core Raspberry Pi since the device only needs to spawn processes. Meanwhile, the workers are three Raspberry Pi 3Bs with four logical processors, which have a better computation process than the master and are suitable for capturing process. The database is also assigned to the last worker to handle data compilation and storage [26], [30], [31]. Since the master device is headless, remote access is required to start the capture process.

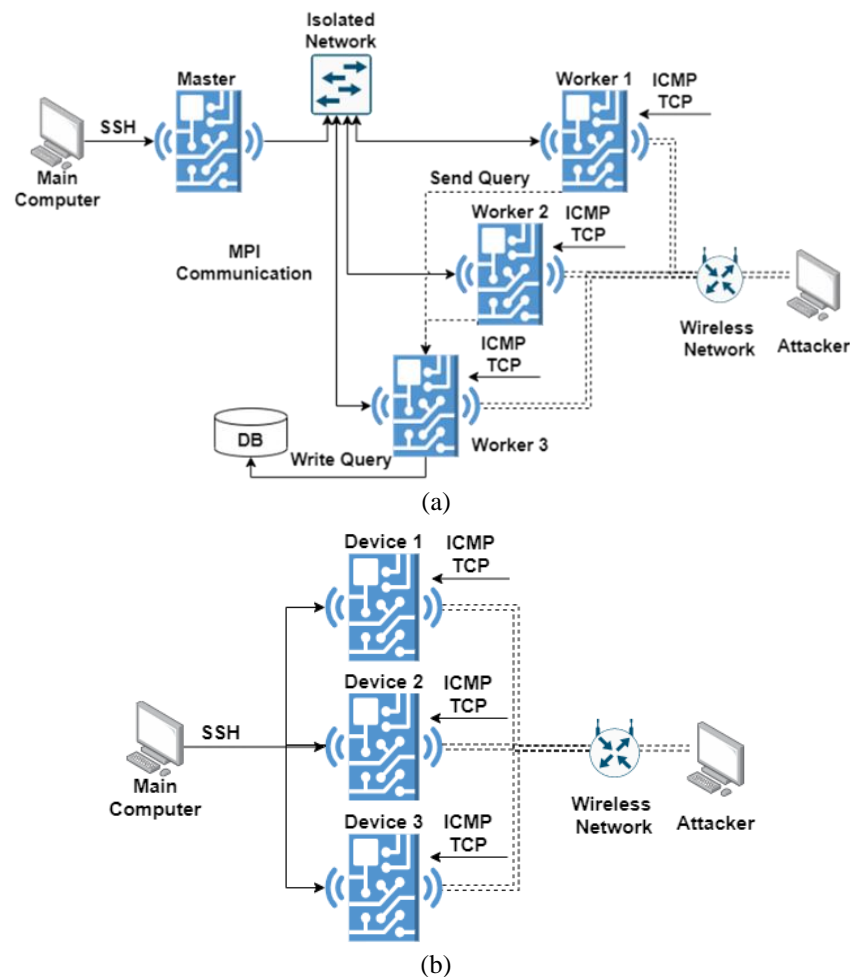


Figure 5. Topology comparison where (a) is the proposed model, and (b) is the typical model

The next step is to build links and communication between devices by connecting all boards to an isolated local area network to ensure undisturbed master-to-workers and workers-to-database communications. After the internal network setup, the three worker devices and the attacker's computer are connected to a wireless network with a maximum speed of up to 4.6 MB/second for single-channel communication. Therefore, when used concurrently between devices, each can utilize the channel with an average speed of 1.17 MB/second. The typical model does not have a controller to manage the devices, hence, each capture process acts independently without interference. Figure 6 is the physical installation of the proposed model:

According to Figure 6, the proposed model consists of a 5-ports switch, a wireless router, and the proposed model. The model has one master node on the rightmost and three worker nodes on the left. Since the model only needs three worker nodes, this study leaves the leftmost node turned off and not connected to the local area network (LAN) network. Each node connects to the switch and forms an isolated LAN network through a wired cable. Besides that, the nodes connect to the wireless network through a wireless router on the right side like Figure 5(a).

After the topology setup, the next step is the configuration of the experiment scenarios, which explains the flooding attacks and capturing process. This study used two protocols for flooding attacks,

namely ICMP and TCP, with a simple shell script written to spawn flooding attacks simultaneously toward the target. The script can shorten the attack spawning time with the help of the experiment process. The first scenario is to test the proposed models with flood attacks, while the second examines the previous model. The result from each scenario contains benchmark data for further analysis.

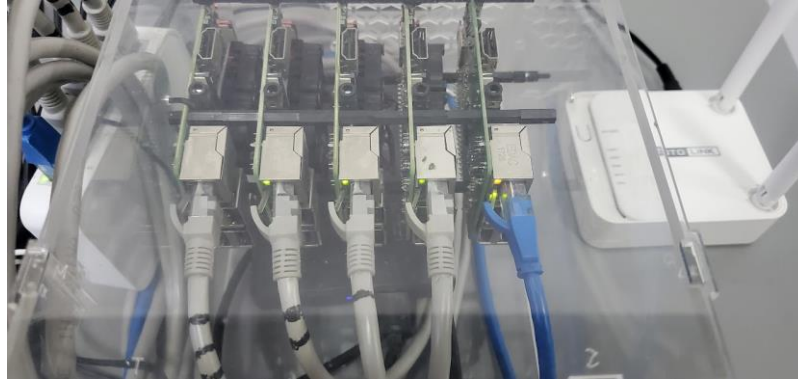


Figure 6. The USS usage comparison

#### 4. RESULTS AND DISCUSSION

The experiment successfully gathered 3,072 packets for each model, consisting of three benchmark indicators, with the collection comprising several thousand rows used to calculate the average per field. Table 2 contains the average calculation of the sample data separated into two columns, namely scalable and non-scalable models for the proposed and typical models.

Table 2. The Benchmark result comparison summary

No	Average Non-Scalable			Average Scalable		
	USS	CPU	Time	USS	CPU	Time
1	13.75	0.00	3.33	17.81	0.00	3.00
2	13.94	15.50	2.67	18.13	6.66	2.67
3	14.00	11.33	3.00	18.14	9.29	3.00
4	14.00	9.20	3.33	18.26	6.85	2.33
5	13.97	14.31	3.33	18.24	9.44	3.00
6	14.01	11.50	3.33	18.43	6.76	3.33
7	14.05	15.27	3.00	18.39	7.77	2.67
8	14.02	14.99	3.00	18.44	6.80	2.33
9	14.08	14.79	4.00	18.46	9.66	3.00
10	14.07	13.18	4.00	18.51	6.51	3.00

Table 2 shows that each model utilized hardware resources differently. The USS indicator represented the actual use of the running model during the capture process at scalable and non-scalable values of 18.82 and 14.43 MB USS memory. The CPU usage showed different results compared to previous indicators, with scalable and non-scalable values of 3.07% and 3.22%. In terms of execution time, the scalable and non-scalable models captured packets within 3.07 and 3.22 ms. Figure 7 presents the USS usage gap between both models.

Figure 7 shows that the scalable and non-scalable models consumed an average of 18.82 and 14.43 MB of memory. There is an extra 4.39MB memory usage of the scalable model to capture the traffic, despite consuming 30.44% more memory than the non-scalable. This extra memory will not burden the system since most IoT device has quite a large memory to run many processes. Meanwhile, Figure 8 shows the CPU usage comparison models.

Unlike the USS usage counterpart, the CPU usage between models in this experiment is similar. Figure 8 shows that the scalable and non-scalable models only used 10.79% and 12.65% of average CPU usage. There is a 1.85% gap between both models, but the scalable result is lower because it consumed 14.66% less CPU, which is more efficient for the IoT device. Furthermore, high CPU usage produces higher temperatures capable of damaging the board without cooling equipment. Both benchmarks affect the execution time of the model, Figure 9 shows the execution time comparison between both models.



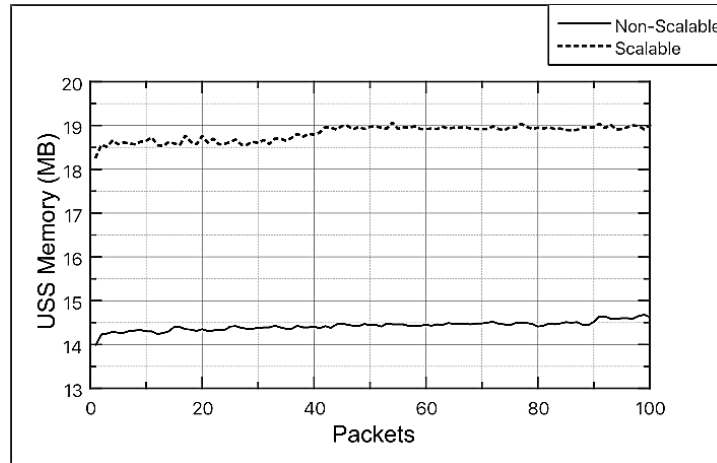


Figure 7. The USS usage comparison

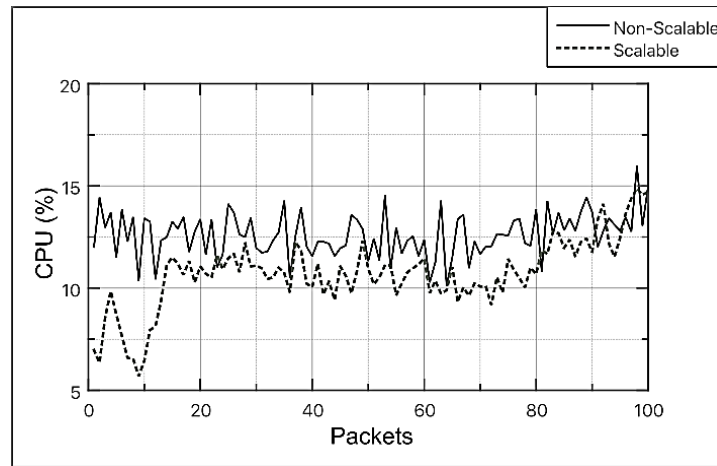


Figure 8. The CPU usage comparison

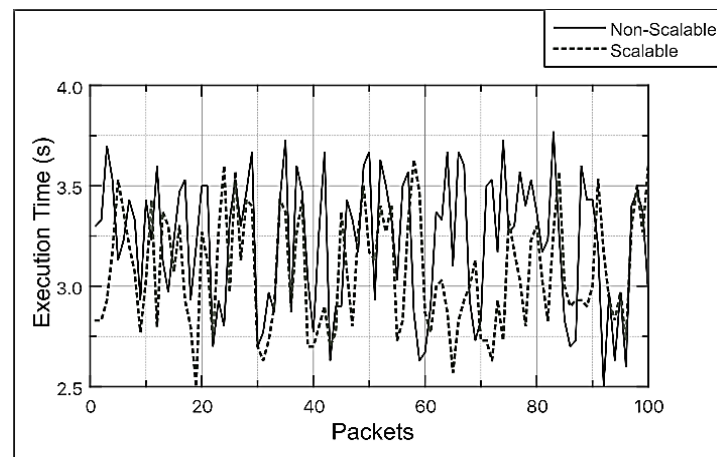


Figure 9. The CPU usage comparison

The execution time benchmark refers to how fast the device captures the packet in milliseconds. According to the benchmark data in Figure 9, the scalable and non-scalable models only need averages of 3.07 and 3.22 ms to capture data packets. Based on this data, the scalable model executes 4.62% faster than



the non-scalable, which helps the user to obtain the data as much as possible. Since the execution time's data is already obtained, the capture rate of both models can be calculated. The capture rate of the non-scalable model is shown in (3).

$$\text{Capture Rate (non)} = \frac{1024 \text{ packets}}{\left(\frac{3300 \text{ ms}}{1000}\right)} = \frac{1024 \text{ packets}}{3.3 \text{ sec}} = 310 \text{ packets/sec} \quad (3)$$

Meanwhile, the result of the scalable model is shown in (4).

$$\text{Capture Rate (scalable)} = \frac{1024 \text{ packets}}{\left(\frac{3147 \text{ ms}}{1000}\right)} = \frac{1024 \text{ packets}}{3.147 \text{ sec}} = 325 \text{ packets/sec} \quad (4)$$

According to the result, the scalable model can capture 15 more packets than the non-scalable due to its ability to increase the capture rate by 5%. This means that the proposed model provides better performance than the former. Therefore, this study utilized an independent left-tailed test to determine and give more evidence of whether the scalable is more efficient.

The independent left-tailed test results in Table 1 will decide whether the hypothesis for each indicator is accepted or rejected. Table 3 shows that the USS indicator hypothesis is accepted in the scalable model because it has more memory than the non-scalable due to the storage of cluster information MPI. The scalable model holds cluster information to recognize and communicate with the rest of the devices. Every process running in each device of the scalable model carried the device's sequence (rank), the size of the total available logical processors, and the hostname. Meanwhile, the non-scalable model did not carry this kind of information in its process because it lacked adequate memory. The CPU and exec time benchmark result also rejected the null hypothesis.

Since the  $t$ -statistic is lower than the  $t$ -critical one tail test, the alternate hypothesis process was used. The scalable model utilized less CPU, less power, and a faster packet capture process than the non-scalable. Since the scalable model spreads its process to several allocated logical processors, the needed processing power is lower. The non-scalable model relied on the device's system to allocate the process to available logical processors. In terms of execution time, the non-scalable model runs a bit slower than the scalable. The system processor management might affect the non-scalable execution time to run a specific process in a particular logical processor. Based on the hypotheses' overall results, this study concluded that the scalable model is more efficient in processing power than the non-scalable. Although the scalable carried more data in its process, it held necessary cluster information to ensure the master and workers' devices communicate properly.

Table 3. The independent left one-tailed test result

Indicators	t Statistic	t Critical One-Tail	Result
USS	1088,154875	-1,64634450	Accepted
CPU Usage	-17,16346244	-1,64634450	Rejected
Exec Time	-6,888482178	-1,64634450	Rejected

## 5. CONCLUSION

The experiment results for both scalable and non-scalable models showed that the scalable model utilized 30.44% more USS than the non-scalable. However, the scalable model held necessary cluster information in its process to ensure master-to-workers communications by using 14.66% less CPU. In terms of execution time, the scalable model executes 3.63% faster than the non-scalable, which benefits users due to its ability to capture and analyze traffic without hindering the board's performance. Besides, users can obtain more data quickly from different networks than in the non-scalable model.

This model allowed the network interface to work separately under a single master device command. At least two single-board computers capable of processing are required to make this model work properly. Despite its ability to capture more data, some weaknesses are considered, such as the need for additional swap space to capture more data, vital communication between devices, and disruption due to accidental disconnection. The network interface is limited to one logical processor's core only. Further studies need to improve more capturing capability, such as writing the proposed method with a programming language close to machine language (C++), implementing in more modern IoT devices, expanding the capture varieties to many network protocols, adding intrusion detection capability for a more secure network, and multi wireless network interface in each device management to capture more significant traffic.

## ACKNOWLEDGEMENTS

The authors are grateful to the Education and Culture Ministry Republic Indonesia for supporting this study through the Grant Research World Class Professor at 2021.





## REFERENCES

- [1] A. Schroeder, A. Z. Bigdeli, C. G. Zarco, and T. Baines, "Capturing the benefits of industry 4.0: a business network perspective," *Production Planning & Control*, vol. 30, no. 16, pp. 1305–1321, Dec. 2019, doi: 10.1080/09537287.2019.1612111.
- [2] A. Chowdhury and S. A. Raut, "Benefits, challenges, and opportunities in adoption of industrial IoT," *International Journal of Computational Intelligence & IoT*, vol. 2, no. 4, 2019.
- [3] I. Butun, P. Osterberg, and H. Song, "Security of the internet of things: Vulnerabilities, attacks, and countermeasures," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 616–644, 2020, doi: 10.1109/COMST.2019.2953364.
- [4] H. Shahriar and S. Nimmagadda, "Network intrusion detection for TCP/IP packets with machine learning techniques," in *Machine Intelligence and Big Data Analytics for Cybersecurity Applications*, 2021, pp. 231–247.
- [5] M. A. Jabbar and R. Aluvalu, "Intrusion detection system for the internet of things: A review," *Smart Cities Symposium 2018*, 2018, doi: 10.1049/cp.2018.1419.
- [6] R. Doshi, N. Aphorpe, and N. Feamster, "Machine learning DDoS detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, May 2018, pp. 29–35, doi: 10.1109/SPW.2018.00013.
- [7] J. Uramova, P. Segec, M. Moravcik, J. Papan, T. Mokos, and M. Brodec, "Packet capture infrastructure based on moloch," in *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Oct. 2017, pp. 1–7, doi: 10.1109/ICETA.2017.8102538.
- [8] P. Emmerich, M. Pudelko, S. Gallenmuller, and G. Carle, "FlowScope: Efficient packet capture and storage in 100 Gbit/s networks," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, Jun. 2017, pp. 1–9, doi: 10.23919/IFIPNetworking.2017.8264852.
- [9] J. Li, C. Wu, J. Ye, J. Ding, Q. Fu, and J. Huang, "The comparison and verification of some efficient packet capture and processing technologies," in *2019 IEEE International Conference on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCCom/CyberSciTech)*, Aug. 2019, pp. 967–973, doi: 10.1109/DASC/PiCom/CBDCCom/CyberSciTech.2019.00177.
- [10] R. Varatharajan, A. P. Preethi, G. Manogaran, P. M. Kumar, and R. Sundarasekar, "Stealthy attack detection in multi-channel multi-radio wireless networks," *Multimedia Tools and Applications*, vol. 77, no. 14, pp. 18503–18526, Jul. 2018, doi: 10.1007/s11042-018-5866-z.
- [11] A. Xiaoguang and L. Xiaofan, "Packet capture and protocol analysis based on winpcap," in *2016 International Conference on Robots & Intelligent System (ICRIS)*, Aug. 2016, pp. 272–275, doi: 10.1109/ICRIS.2016.55.
- [12] D. Bastos, M. Shackleton, and F. El-Moussa, "Internet of things: A survey of technologies and security risks in smart home and city environments," *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, 2018, doi: 10.1049/cp.2018.0030.
- [13] D. D. Khudhur and M. S. Croock, "Physical cyber-security algorithm for wireless sensor networks," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 19, no. 4, pp. 1177–1184, Aug. 2021, doi: 10.12928/telkommika.v19i4.18464.
- [14] V. Simadiputra and N. Surantha, "Rasefiberry: Secure and efficient Raspberry-Pi based gateway for smarthome IoT architecture," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 2, pp. 1035–1045, Apr. 2021, doi: 10.11591/eei.v10i2.2741.
- [15] S. Kurgalin and S. Borzunov, "Fundamentals of parallel computing," in *A Practical Approach to High-Performance Computing*, Cham: Springer International Publishing, 2019, pp. 17–35.
- [16] M. Brinskiy, M. Lubin, and J. Dinan, "MPI-3 shared memory programming introduction," in *High Performance Parallelism Pearls*, Elsevier, 2015, pp. 305–319.
- [17] F. Desprez *et al.*, Eds., *Euro-par 2016: Parallel processing workshops*, vol. 10104. Cham: Springer International Publishing, 2017.
- [18] R. Smith, "Performance of MPI codes written in python with numPy and mpi4py," in *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*, Nov. 2016, pp. 45–51, doi: 10.1109/PyHPC.2016.010.
- [19] S. Kurgalin and S. Borzunov, "The MPI technology," in *A Practical Approach to High-Performance Computing*, Cham: Springer International Publishing, 2019, pp. 37–66.
- [20] G. Dorr, D. Hagen, B. Laskowski, E. Steinmetz, and D. Vo, "Introduction to parallel processing with eight node Raspberry Pi cluster," in *Midwest Instruction and Computing Symposium (MICS)*, The University of Wisconsin-La Crosse in La Crosse, 2017, pp. 7–8.
- [21] Y. Zheng and P. Marguinaud, "Simulation of the performance and scalability of message passing interface (MPI) communications of atmospheric models running on exascale supercomputers," *Geoscientific Model Development*, vol. 11, no. 8, pp. 3409–3426, Aug. 2018, doi: 10.5194/gmd-11-3409-2018.
- [22] A. A. Ismail, H. S. Hamza, and A. M. Kotb, "Performance evaluation of open source IoT platforms," in *2018 IEEE Global Conference on Internet of Things (GCIoT)*, Dec. 2018, pp. 1–5, doi: 10.1109/GCIoT.2018.8620130.
- [23] M. T. Islam, N. Islam, and M. Al Refat, "Node to node performance evaluation through RYU SDN controller," *Wireless Personal Communications*, vol. 112, no. 1, pp. 555–570, May 2020, doi: 10.1007/s11277-020-07060-4.
- [24] M. Swain, R. Singh, M. F. Hashmi, and A. Gehlot, "Performance analysis of various embedded Linux firmwares for ARM architecture based IoT devices," in *International Conference on Intelligent Computing and Smart Communication 2019*, 2020, pp. 1451–1460.
- [25] S. K. Gutierrez, "A memory consumption benchmark for MPI implementations," Los Alamos, NM (United States), Nov. 2018. doi: 10.2172/1482909.
- [26] I. Alsmadi, S. Khamaiseh, and D. Xu, "Network parallelization in HPC clusters," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec. 2016, pp. 584–589, doi: 10.1109/CSCI.2016.0116.
- [27] J. Kävestad, "Memory analysis tools," in *Fundamentals of Digital Forensics*, Cham: Springer International Publishing, 2020, pp. 217–224.
- [28] A. Jackson, A. Turner, M. Weiland, N. Johnson, O. Perks, and M. Parsons, "Evaluating the Arm ecosystem for high performance computing," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, Jun. 2019, pp. 1–11, doi: 10.1145/3324989.3325722.





- [29] D. Papakyriakou, D. Kottou, and I. Kostouros, "Benchmarking Raspberry Pi 2 beowulf cluster," *International Journal of Computer Applications*, vol. 179, no. 32, pp. 21–27, Apr. 2018, doi: 10.5120/ijca2018916728.
- [30] A. Selinger, K. Rupp, and S. Selberherr, "Evaluation of mobile ARM-based SoCs for high performance computing," *24th High Performance Computing Symposium*, 2016, doi: 10.22360/SpringSim.2016.HPC.022.
- [31] S. McIntosh-Smith, J. Price, T. Deakin, and A. Poenaru, "A performance analysis of the first generation of HPC-optimized Arm processors," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 16, Aug. 2019, doi: 10.1002/cpe.5110.

## BIOGRAPHIES OF AUTHORS







**Kristoko Dwi Hartomo**     completed his doctorate degree on the Doctorate Program of Computer Science, Science Faculty of Gadjah Mada University in 2017. He has been active on research since 2008 until now on the intrusion detection system, spatial data processing and remote sensing. He has published his papers on international journals, has 5 copyrights, and wrote some reference books on remote sensing data analysis and modelling in Indonesia and English. He can be contacted at [kristoko@uksw.edu](mailto:kristoko@uksw.edu).



**April Firman Daru**     is currently studying for a Doctor of Computer Science at the Universitas Kristen Satya Wacana (UKSW) Salatiga, Indonesia. Since 2010 until now he has been a Lecturer in the Informatics Engineering Study Program at the Universitas Semarang (USM). His research interests are in the fields of internet of things, artificial intelligence, attack detection and machine learning. He can be contacted at [firman@usm.ac.id](mailto:firman@usm.ac.id).



**Hindriyanto Dwi Purnomo**     is an Associate Professor at Department of Information Technology, Universitas Kristen Satya Wacana, Indonesia. He received his bachelor's degree in engineering physics, from Gadjah Mada University, Indonesia, in 2005, Magister of Information Technology from The University of Melbourne, Australia in 2009 and Doctor of Philosophy in Industrial and System Engineering from Chung Yuan Christian University, Taiwan, in 2013. He has received several recognitions and awards for his academic achievement. He also has published many articles in reputable journals, conferences, and books. His research interests are in the field of metaheuristics, soft computing, machine learning and deep learning. He can be contacted at [hindriyanto.purnomo@uksw.edu](mailto:hindriyanto.purnomo@uksw.edu).