https://doi.org/10.31891/2307-5732-2025-357-88
УДК 004.75:004.738.5

ЯВОРСЬКИЙ ГЕРМАНН
Національний лісотехнічний університет України
https://orcid.org/0009-0002-3410-5102
e-mail: hermann.yav@gmail.com
ФЛУД ЛЮБОМИР
Національний лісотехнічний університет України
https://orcid.org/0000-0002-8347-4265
e-mail: flud@nltu.edu.ua

СИСТЕМА МОНІТОРИНГУ МЕРЕЖЕВОГО ТРАФІКУ У KUBERNETES КЛАСТЕРІ

У роботі досліджено методи та інструменти аналізу та моніторингу мережевого трафіку у Kubernetes кластері на основі технології Berkeley Packet Filtering. У рамках дослідження спроектовано та розроблено систему захоплення та обробки мережевих пакетів на основі технології Berkeley Packet Filtering та збереження оброблених даних у базі даних Prometheus. Для розробки системи було використано мову програмування Rust з використанням функцій з бібліотеки pcap для захоплення пакетів за допомогою технології BPF.

Ключові слова: мережевий трафік, моніторинг трафіку, Kubernetes, аналіз трафіку.

YAVORSKYI HERMANN
FLUD LIUBOMYR
Ukrainian National Forestry University

NETWORK TRAFFIC MONITORING SYSTEM IN A KUBERNETES CLUSTER

The efficient operation of modern cloud infrastructures relies on effective network traffic monitoring and management. Kubernetes, the leading platform for orchestrating containerized applications, automates deployment and scaling but lacks built-in tools for detailed network traffic analysis. This paper presents methods and tools for capturing and monitoring network traffic within Kubernetes clusters, using Berkeley Packet Filtering (BPF) technology. As part of the study, a system was designed and implemented to capture network packets based on BPF and store the processed data in a Prometheus database for analysis.

The system was developed in the Rust programming language, chosen for its high performance and memory safety. Packet capture was implemented using the pcap library, which enables efficient interaction with network traffic at the kernel level. The monitoring solution focuses on traffic between Kubernetes nodes, traffic between pods located on different nodes, and external traffic entering or leaving the cluster, while internal traffic within a single node is excluded to optimize system performance. Traffic classification is based on the source and destination IP addresses of captured packets. Traffic is categorized as internal if associated with node or pod IP ranges, and external otherwise. Metrics such as packet counts and total traffic volume are collected and structured for Prometheus, allowing further visualization and analysis using standard tools like Grafana. The monitoring system is deployed using Kubernetes DaemonSets, ensuring the capture application runs on each node automatically. This approach enables full coverage without manual intervention when scaling the cluster. Thanks to the use of Rust's asynchronous programming model with Tokio, the system can efficiently handle dynamic cluster changes while maintaining low resource consumption.

The result is a scalable, lightweight traffic monitoring solution that provides valuable insights into Kubernetes network behavior.

Keywords: network traffic, traffic monitoring, Kubernetes, traffic analysis.

Стаття надійшла до редакції / Received 01.07.2025
Прийнята до друку / Accepted 15.08.2025

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

У полі Інформаційних Технологій багато рішень, навіть тих, що вирішують невеликі задачі, потребують значної допоміжної інфраструктури для функціонування. Наприклад, для роботи інтернет-магазину потрібні системи керування товарами, авторизації користувачів (або навіть продавців, як у маркетплейсах, наприклад, Rozetka), обробки платежів і замовлень, сховище даних та комунікація з сервісами доставки. Це призвело до популяризації мікросервісної архітектури на базі хмарних обчислень.

Усі ці системи потрібно розгорнути та налаштувати їхню взаємодію. Цю задачу вирішує Kubernetes - інструмент з відкритим вихідним кодом, який автоматизує розгортання, масштабування та керування контейнеризованими сервісами. Він об'єднує сервери в кластер та забезпечує абстракції для взаємодії з сервісами, що розгортаються. Згідно з опитуванням Cloud Native Computing Foundation, популярність Kubernetes зросла з 58% у 2018 році до 83% у 2020 році [1].

Kubernetes — це високоефективна платформа, що дозволяє вирішувати основні завдання з адміністрування різноманітних сервісів, які працюють у кластерному середовищі, зокрема забезпечує контроль мережевого трафіку за допомогою ресурсів типу NetworkPolicy [2]. Однак у стандартній конфігурації Kubernetes не передбачає засобів для моніторингу або аналізу мережевого трафіку, зокрема для визначення обсягів даних, що виходять за межі внутрішньої мережі кластера та спрямовуються до зовнішніх ресурсів, зокрема в Інтернет.

Аналіз досліджень та публікацій

Управління мережевими з'єднаннями в Kubernetes є однією з ключових технічних складових, що суттєво впливає на стабільність та ефективність функціонування кластера. На відміну від традиційних мережевих архітектур, Kubernetes передбачає високий рівень динаміки: постійне масштабування застосунків, переміщення Pod між вузлами, а також швидкі зміни в конфігурації сервісів. У таких умовах реалізація ефективної та керованої мережі потребує використання плагінів Container Networking Interface (CNI), які забезпечують маршрутизацію, сегментацію, контроль доступу та взаємодію між компонентами. Серед найбільш поширених CNI-рішень, що використовуються у хмарних середовищах (зокрема, в Azure Kubernetes Service), можна виділити kubenet, Flannel та Calico [3, 4], які реалізують різні підходи до побудови мережевої взаємодії в межах кластерів.

Попри широке застосування Kubernetes, інструменти для детального моніторингу мережевого трафіку в межах кластерів залишаються обмеженими. Наявні рішення здебільшого орієнтовані на облік ресурсів або базову візуалізацію метрик. Інтерес викликає проєкт Coroot (https://coroot.com/), який інтегрується з Kubernetes-ресурсами та реалізує збір, обробку і візуалізацію мережевого трафіку. Його функціонал дозволяє виявляти проблемні зони взаємодії компонентів та будувати карту комунікацій всередині кластера [5].

Недоліком зазначеного рішення є потреба в установці додаткових компонентів, що збільшує навантаження на інфраструктуру, створює нові залежності та ускладнює експлуатацію. З огляду на це, в даній роботі пропонується підхід до побудови спостереження за мережевим трафіком з мінімальним впливом на середовище. Розроблена система забезпечує інтеграцію зі стандартними інструментами спостереження (Prometheus, Grafana) та використовує технології BPF, Rust і Helm, не потребуючи складної зовнішньої інфраструктури.

Формулювання цілей статті

Суть дослідження полягає у розгляді методів та засобів для аналізу й контролю мережевого трафіку всередині Kubernetes-кластерів. Основним об'єктом вивчення є система, яка здійснює моніторинг мережевої активності у середовищі Kubernetes.

Головна мета цієї роботи полягає у створенні та впровадженні інструментарію для відстеження мережевого трафіку в Kubernetes кластері. Для досягнення цього використано сучасні підходи до розгортання подібних рішень, зокрема застосування мови Rust, технологій BPF (pcap), DaemonSet та збору метрик через Prometheus.

Виклад основного матеріалу

Головним кроком цього дослідження є визначення основних типів мережевого трафіку, які становлять інтерес для даної роботи. Зокрема, досліджується можливість спостереження за такими видами трафіку: обмін даними між окремими вузлами (Nodes), взаємодія між різними Pod-об'єктами, а також аналіз зовнішнього трафіку - як вихідного з Kubernetes-кластера в Інтернет, так і вхідного, що потрапляє в кластер ззовні. Виходячи з цього, було виділено такі групи мережевого трафіку:

- Node-to-Node - передача даних між окремими вузлами кластера;
- Node-to-External Network - трафік, що спрямовується з вузлів у зовнішні мережі;
- External Network-to-Node - потік даних з Інтернету або інших зовнішніх ресурсів до вузла;
- Pod-to-Pod між вузлами - взаємодія контейнеризованих застосунків (Pods), які розміщені на різних вузлах.

Далі детальніше розглянемо стандартну мережеву архітектуру Kubernetes [3, 6].

Уся взаємодія між контейнеризованими застосунками, яка виходить за межі одного вузла, обов'язково спрямовується через мережевий інтерфейс eth0 на відповідному вузлі. Варто зазначити, що обмін даними між Pod-ами, які розташовані на одному й тому ж вузлі, не використовує цей інтерфейс [6]. Це зменшує потенційне навантаження на систему, адже такий тип трафіку не є цікавим в рамках цього дослідження.

Для визначення типу мережевого трафіку планується здійснювати аналіз IP-адреси відправника та IP-адреси одержувача кожного пакета. У разі, якщо IP-адреса відправника співпадає з IP вузла або входить до діапазону Pod CIDR [7, 8] цього вузла, це означає, що пакет був сформований всередині кластера. Аналогічно, якщо IP-адреса одержувача відповідає IP вузла чи знаходиться у межах його Pod CIDR, одержувач пакета розташований у кластері. Якщо жодна з цих умов не виконується, можна вважати, що пакет або надходить із зовнішньої мережі, або спрямований назовні.

Облік кількості та розміру мережевих пакетів здійснюється з урахуванням IP-адрес відправника й одержувача. Це надає змогу ретельно дослідити, які вузли або Pod генерують найбільші обсяги трафіку, а також які з них приймають найбільше даних. Всі метрики поділяються на дві категорії: для внутрішнього й зовнішнього трафіку. Внутрішній трафік охоплює передачу даних між вузлами та Pod всередині самого кластера, а зовнішній - взаємодію кластера із зовнішніми мережами.

Структура даних, яка використовується для збору цих показників, добре сумісна з метриками Prometheus [9]. Під метрикою мається на увазі одиниця виміру, яка дає змогу контролювати певний аспект функціонування системи. У нашому випадку метрики фіксуватимуть кількість і розмір мережевих пакетів, IP-адреси відправника й одержувача, а також часові характеристики.

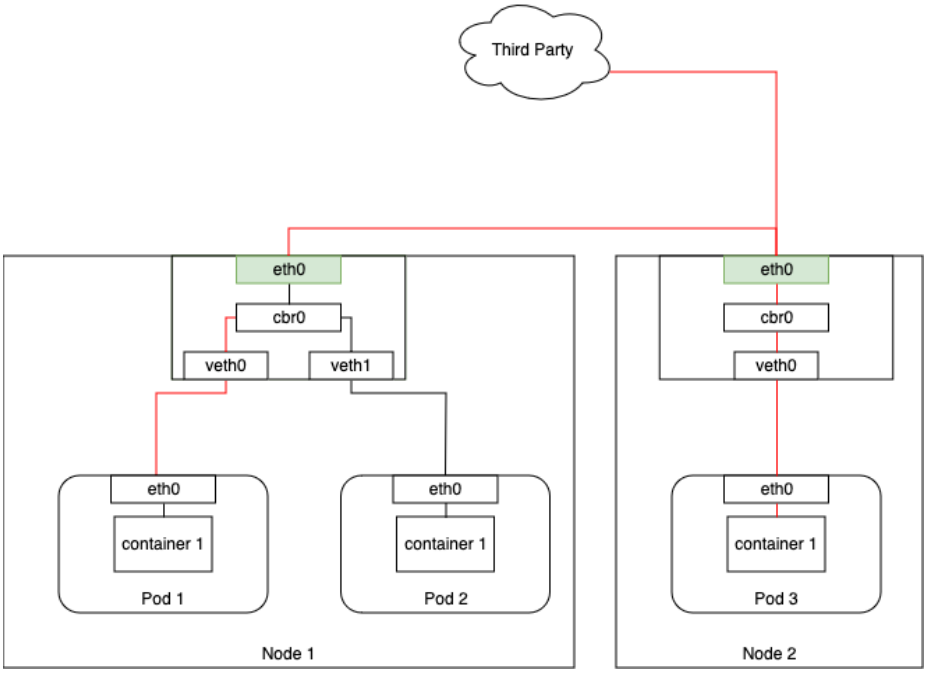

Рис. 1. Схема обміну мережевими даними між вузлами та поза межами Kubernetes-кластера

Для створення проекту була використана мова програмування Rust. Rust є сучасною системною мовою, яка поєднує високу продуктивність із надійністю керування пам'яттю. Рішення передбачає обробку багатьох подій одночасно: додавання/видалення вузла у кластері, зміна топології і в той же час захоплення пакетів. Для оптимального результату необхідно обробляти ці події одночасно, щоб коли пакет, який направлений до вузла, що щойно був створений, був захоплений, система вже мала інформацію про новий вузол. Для цього використано бібліотеку Tokio, що надає інструменти для побудови ефективних та масштабованих асинхронних додатків. Для захоплення пакетів використано бібліотеку pcap, що імплементовує технологію Berkeley Packet Filtering (BPF). BPF та розширення цієї технології eBPF дозволяє захоплювати мережеві пакети з мінімальними наслідками для швидкодії системи [10].

Отже, ціль системи це захоплювати пакети на інтерфейсі eth0 кожного вузла у кластері, та визначати його source та destination IP-аддреси. І в подальшому зберігати дані про цей пакет як метрики. Для цього застосунок має підтримувати актуальний список вузлів, та CIDR об'єктів Pod на цих вузлах.

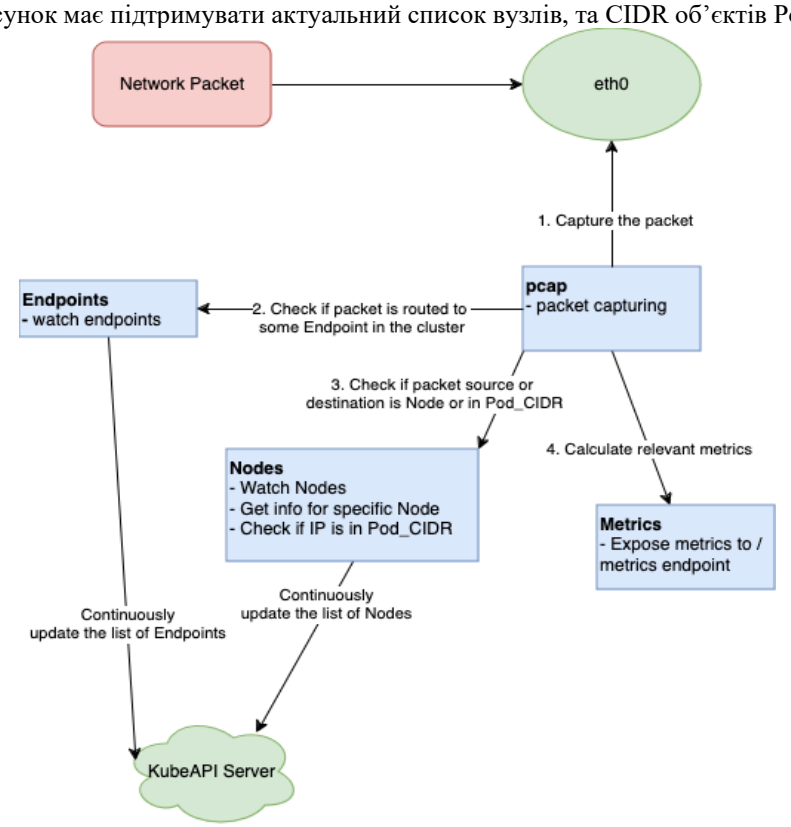

Рис. 2. Схема роботи застосунку для фільтрування мережевого трафіку

Для розгортання системи, що має аналізувати мережевий трафік на всіх вузлах у кластері, було використано об'єкт DaemonSet. DaemonSet у Kubernetes дозволяє запускати Pod із застосунком на кожному вузлі у кластері. Він забезпечує автоматичне розгортання Pod на всіх вузлах, що є корисним для сценаріїв, де необхідний доступ до кожного вузла, наприклад, при зборі журналів, моніторингу чи виконанні мережевих операцій. DaemonSet гарантує, що при додаванні або видаленні вузла відповідні Pod автоматично створюються або видаляються. Схема розгортання системи у кластері.

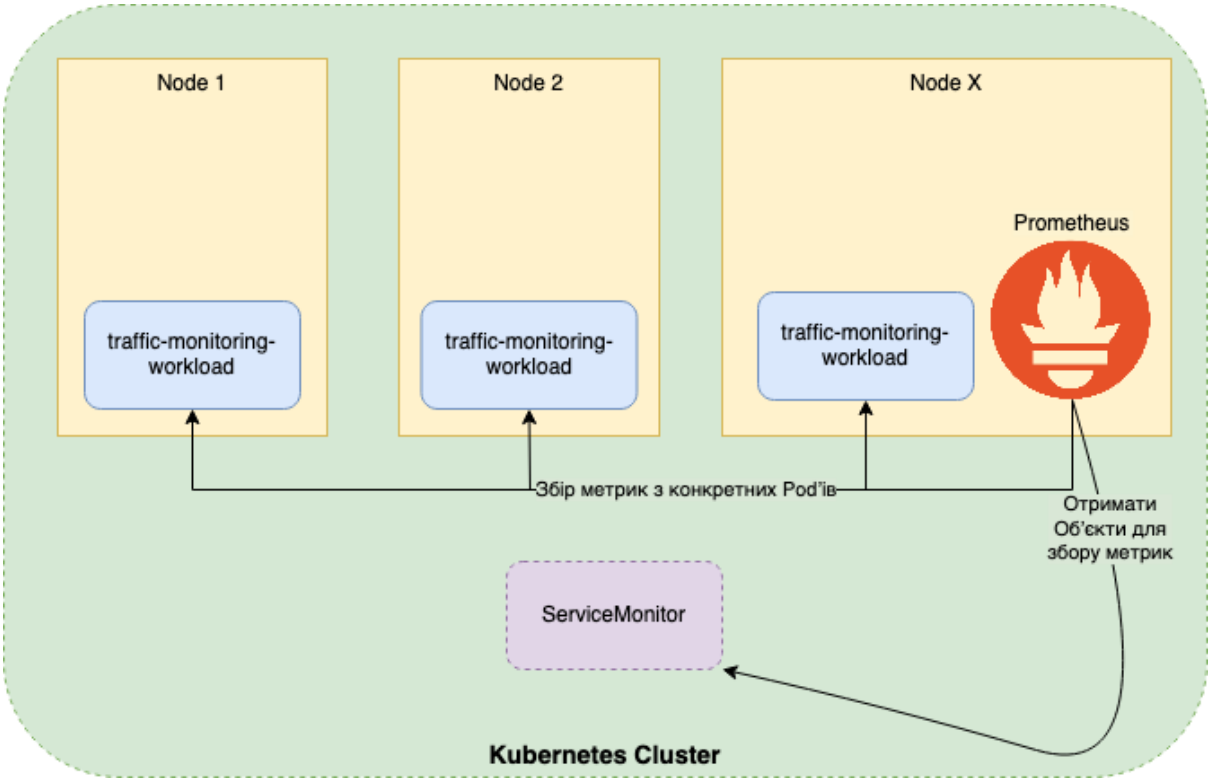

Рис. 3. Архітектура розгортання компонентів проекту у Kubernetes-кластері

Результатом роботи системи, тобто обробки мережевих пакетів, є набір відповідних метрик у Prometheus.

Рис. 4. Графічне представлення метрик у Prometheus Dashboard

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Підсумком виконаної роботи стало створення системи, яка здатна перехоплювати мережеві пакети, обробляти їх і формувати відповідні метрики у вигляді, що підтримується Prometheus. Розроблене рішення дозволяє оцінювати обсяги трафіку між різними вузлами у кластері, а також той, що покидає його межі. Впроваджена система надає важливий функціонал, якого бракує у Kubernetes за замовчуванням, і може бути використана для моніторингу обсягу трафіку, виявлення аномальних ситуацій, оцінки ефективності роботи застосунків та підвищення оптимальності мережевої інфраструктури. Завдяки високій продуктивності, хорошій масштабованості та можливості інтеграції з іншими інструментами, система має потенціал для подальшого розвитку й адаптації до різних сценаріїв використання.

Література

1. Cloud Native Computing Foundation. CNCF Survey Report 2020 (с. 8). CNCF, 2020. https://www.cncf.io/wp-content/uploads/2020/12/CNCF_Survey_Report_2020.pdf.
2. Budigiri, Gerald & Baumann, Christoph & Muhlberg, Jan & Truyen, Eddy & Joosen, Wouter. (2021). Network Policies in Kubernetes: Performance Evaluation and Security Analysis. 407-412. 10.1109/EuCNC/6GSummit51104.2021.9482526. (с. 3-4)
3. The Kubernetes Authors. "The Kubernetes Network Model." // Документація Kubernetes. – [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/cluster-administration/networking/#the-kubernetes-network-model.
4. AKS documentation. // Документація Azure. – [Електронний ресурс]. – Режим доступу: https://docs.azure.cn/en-us/aks/configure-kubenet.
5. Coroot Engineering. Using eBPF and predefined inspections to minimize “observability tax”. // Блог Coroot. – [Електронний ресурс]. – Режим доступу: https://coroot.com/blog/engineering/minimizing-observability-tax/.
6. Sumeet, Kumar. "Understanding Kubernetes Networking: Part 2." Medium, September 20, 2019. [Електронний ресурс]. – Режим доступу: https://medium.com/microsoftazure/understanding-kubernetes-networking-part-2-4b7aa58919bb.
7. Fuller, V.. (2006). Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan.
8. IBM. "Kubernetes Network Model." // Документація IBM. – [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/cloud-private/3.2.x?topic=networking-kubernetes-network-model – (Дата звернення 11.09.2024р.)
9. Jani, Yash. (2024). Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions. Journal of Artificial Intelligence, Machine Learning and Data Science. 2. 848-852. 10.51219/JAIMLD/yash-jani/206. (с. 2-3)
10. Scholz, Dominik & Raumer, Daniel & Emmerich, Paul & Kurtz, Alexander & Lesiak, Krzysztof & Carle, Georg. (2018). Performance Implications of Packet Filtering with Linux eBPF. 209-217. 10.1109/ITC30.2018.00039.

References

1. Cloud Native Computing Foundation. CNCF Survey Report 2020 (p. 8). CNCF, 2020. [Electronic resource]. – Available at: https://www.cncf.io/wp-content/uploads/2020/12/CNCF_Survey_Report_2020.pdf.
2. Budigiri, Gerald & Baumann, Christoph & Muhlberg, Jan & Truyen, Eddy & Joosen, Wouter. (2021). Network Policies in Kubernetes: Performance Evaluation and Security Analysis. pp. 407–412. 10.1109/EuCNC/6GSummit51104.2021.9482526. (pp. 3–4)
3. The Kubernetes Authors. "The Kubernetes Network Model." // Kubernetes Documentation. – [Electronic resource]. – Available at: https://kubernetes.io/docs/concepts/cluster-administration/networking/#the-kubernetes-network-model.
4. AKS documentation. // Azure Documentation. – [Electronic resource]. – Available at: https://docs.azure.cn/en-us/aks/configure-kubenet.
5. Coroot Engineering. Using eBPF and predefined inspections to minimize “observability tax”. // Coroot Blog. – [Electronic resource]. – Available at: https://coroot.com/blog/engineering/minimizing-observability-tax/.
6. Sumeet, Kumar. "Understanding Kubernetes Networking: Part 2." Medium, September 20, 2019. – [Electronic resource]. – Available at: https://medium.com/microsoftazure/understanding-kubernetes-networking-part-2-4b7aa58919bb.
7. Fuller, V. (2006). Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan.
8. IBM. "Kubernetes Network Model." // IBM Documentation. – [Electronic resource]. – Available at: https://www.ibm.com/docs/en/cloud-private/3.2.x?topic=networking-kubernetes-network-model – (Accessed on 11.09.2024).
9. Jani, Yash. (2024). Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions. Journal of Artificial Intelligence, Machine Learning and Data Science, 2, pp. 848–852. 10.51219/JAIMLD/yash-jani/206. (pp. 2–3)
10. Scholz, Dominik & Raumer, Daniel & Emmerich, Paul & Kurtz, Alexander & Lesiak, Krzysztof & Carle, Georg. (2018). Performance Implications of Packet Filtering with Linux eBPF. pp. 209–217. 10.1109/ITC30.2018.00039.