СЕТИ И СИСТЕМЫ СВЯЗИ

# Способ обработки и фильтрации сетевого трафика в системе управления сетевым сегментированием

**И.Г. Бужин,** Московский технический университет связи и информатики (МТУСИ), доцент, к.т.н.; i.g.buzhin@mtuci.ru
**Э.А. Гайфутдинов,** МТУСИ, аспирант; e.a.gaifutdinov@mtuci.ru

УДК 654.073.3 DOI: 10.34832/ELSV.2024.57.8.006

**Аннотация.** *В статье исследованы способы обработки и фильтрации трафика в высоконагруженных сегментах мобильных сетей связи. Выполнена экспериментальная проверка эффективности предложенного способа обработки и фильтрации сетевого трафика в системе сетевого сегментирования. Выявлены особенности обработки и фильтрации сетевого трафика в операционной системе Linux, разработан экспериментальный стенд для проверки модифицированного способа фильтрации сетевого трафика при сетевом сегментировании, получены результаты экспериментальной проверки разработанного способа.*

**Ключевые слова:** *межсетевой экран, iptables, eBPF, сетевой трафик, ядро Linux.*

## ВВЕДЕНИЕ

В современных мобильных сетях появляется необходимость в эффективной сегментации ресурсов сети связи для предоставления сервисов и услуг [1]. Одним из эффективных методов реализации сегментирования на сети является межсетевое экранирование, которое осуществляет контроль и фильтрацию проходящего через него сетевого трафика в соответствии с заданными правилами [2]. При управлении сегментированием сетевых ресурсов важным аспектом является реализация правил обработки и фильтрации пакетов, которые должны выполняться за наименьшее время.

В современных высокоскоростных сетях задача фильтрации трафика приобретает всё большее значение [3]. С ростом объемов данных и увеличением числа угроз обеспечение безопасности сетевого взаимодействия становится первостепенной задачей для систем управления сегментированием сети [4]. Фильтрация трафика позволяет не только блокировать потенциально опасные данные, но и управлять доступом к ресурсам, оптимизировать производительность сети и соблюдать установленные политики безопасности.

Основные задачи фильтрации трафика:

1) безопасность – фильтрация способствует блокированию потока данных, который может представлять угрозу для безопасности сети, например, вредоносные программы, атаки на уровне сетевого протокола и другие формы киберугроз;

2) управление доступом – с помощью фильтрации можно определить, кто и как может взаимодействовать с ресурсами сети (правами доступа, блокировкой конкретных IP-адресов или сетей);

3) оптимизация производительности – фильтрация трафика может использоваться для управления пропускной способностью, приоритетами и ограничением использования ресурсов сети для оптимизации производительности;

4) соблюдение политик и стандартов – администраторы могут устанавливать правила фильтрации для соблюдения политик безопасности, стандартов сетевой безопасности и законодательных требований;

5) управление и балансировка сетевого трафика – алгоритмы фильтрации могут быть настроены для распределения (балансировки) нагрузки сетевого трафика между сетевым узлами с целью повышения отказоустойчивости сети;

6) отладка и мониторинг – фильтрация позволяет администраторам анализировать сетевой трафик для устранения проблем, мониторинга производительности и выявления аномалий.

К средствам фильтрации трафика можно отнести:

- брандмауэры – основной инструмент для фильтрации трафика, который может работать на уровне сетевого стека (например, iptables, nftables);
- прокси-серверы – используются для фильтрации и аутентификации веб-трафика;
- IDS/IPS-системы – системы обнаружения и предотвращения вторжений предоставляют возможности фильтрации на уровне анализа поведения сетевого трафика;

- системы контроля доступа (ACL) – устанавливают правила для управления доступом к ресурсам сети;
- средства управления трафиком (QoS) – используются для оптимизации производительности и приоритизации трафика.

Фильтрация трафика в операционной системе (ОС) Linux реализуется с использованием различных инструментов, таких как iptables, IPSets, eBPF, DPDK и многих других, которые позволяют создавать правила, определяющие, какой трафик разрешен или запрещен на устройстве. Эти правила могут использовать различные параметры, такие как IP-адреса, порты, протоколы и т.д. Однако указанные инструменты и способы фильтрации трафика работают с разной эффективностью. Таким образом, необходимо разработать модифицированный способ фильтрации трафика с меньшей временной сложностью по сравнению с аналогичными способами.

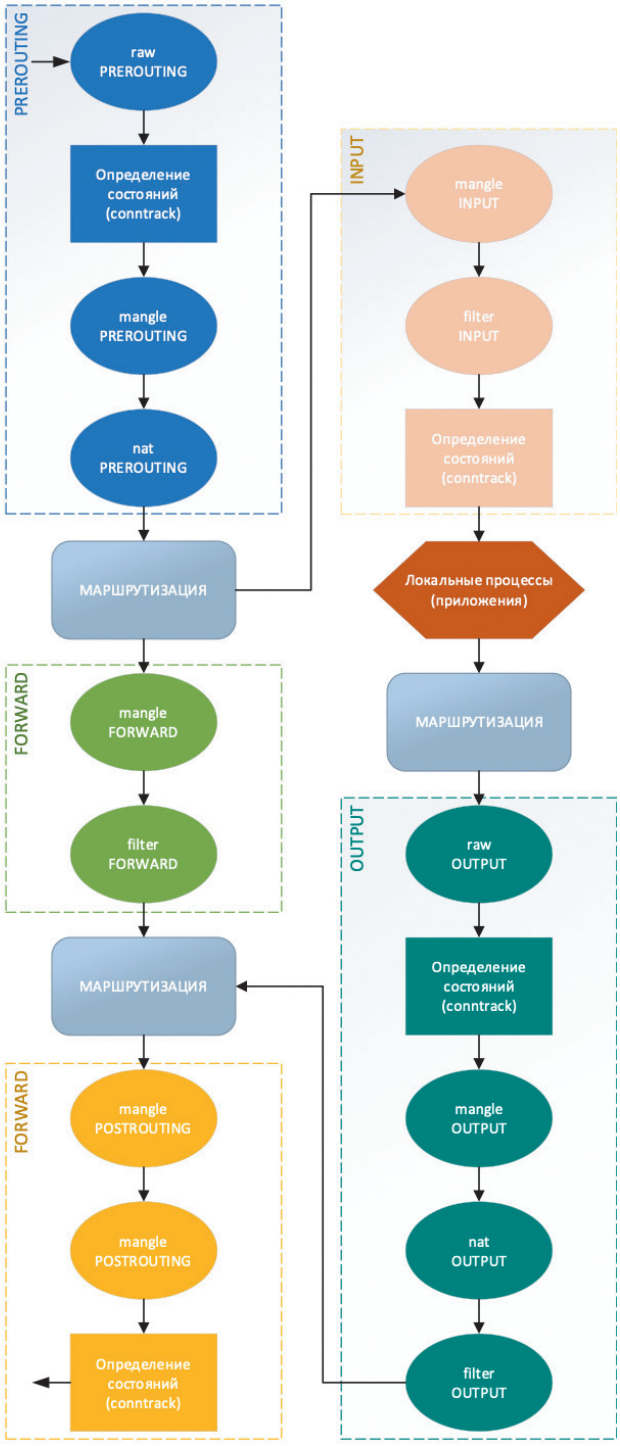
## ФИЛЬТРАЦИЯ СЕТЕВОГО ТРАФИКА С ПОМОЩЬЮ УТИЛИТЫ IPTABLES

Iptables [5] – это утилита командной строки в ОС Linux, предоставляющая интерфейс для настройки правил фильтрации и манипуляций сетевыми пакетами с использованием подсистемы Netfilter в ядре Linux. Для осуществления работы Netfilter необходимы настройки CONFIG_IP_NF_IPTABLES, CONFIG_NETFILTER, CONFIG_IP_NF_NAT, CONFIG_IP_NF_FILTER, CONFIG_BRIDGE_NETFILTER. Также стоит учесть, что минимальная версия ядра для Netfilter – Linux 2.6 [6].

Структурная схема функционирования iptables изображена на рис. 1.

Поступая на сетевой интерфейс, пакеты сначала проходят простые проверки с помощью ядра Linux (к примеру, проверку контрольной суммы) [7]. Затем они отправляются через последовательности цепочек. В начале пакет поступает в цепочку *PREROUTING*, которая является первоначальной. Основываясь на таблице маршрутизации, система определяет, кому принадлежит пакет и переправляет его в нужную цепочку в зависимости от его назначения. Если пакет адресован локальной системе, то перед переходом к локальным процессам он сначала отправляется в цепочку *INPUT*. Пакет, не имеющий отношения к локальной системе и адресованный не ей, попадает в цепочку *FORWARD*. Когда система завершает обработку на локальном уровне, есть возможность сформировать ответ. Ответный пакет отправляется в соответствии со всеми правилами маршрутизации. После генерации он направляется на нужный маршрут и проходит через цепочку *OUTPUT*. После этой цепочки пакет заново проходит полную проверку на соответствие правилам маршрутизации и направляется в цепочку *POSTROUTING*. Если пакет

**Рисунок 1**
Функционирование утилиты iptables




был проходящий, он попадает в *POSTROUTING* после цепочки *FORWARD* [8].

Каждая цепочка всегда включает в себя группу таблиц. Таблица является упорядоченным набором правил, каждое правило включает условие, которому должен соответствовать проходящий пакет, и

определенные действия, которые выполняются над пакетом.

Стоит упомянуть механизм определения состояний (conntrack) [9], который позволяет определить, к какому соединению принадлежит тот или иной пакет. Работает механизм следующим образом: сначала система проводит анализ состояния всех пакетов, кроме тех, которые специально отмечены как NOTRACK (таблица raw), а затем, уже опираясь на это состояние, система определяет принадлежность к тому или иному состоянию.

Существует несколько таких состояний:

- новое соединение (NEW);
- неопределяемое соединение (INVALID);
- установленное соединение (ESTABLISHED);
- относящееся к уже существующему соединению (RELATED).

В различных цепях могут независимо друг от друга существовать пакеты с одинаковыми названиями. К примеру, таблица mangle в цепочке FORWARD не имеет никаких связей с таблицей mangle в цепи POSTROUTING.

Iptables представляет собой мощный инструмент для фильтрации трафика в ОС Linux, предлагает широкий спектр функций, что даёт возможность администраторам конфигурировать правила фильтрации для различных уровней сетевого стека. Также немаловажным преимуществом iptables является поддержка как IPv4, так и IPv6, что обеспечивает возможность создания правил для обоих протоколов. Помимо этого, iptables позволяет администраторам создавать сложные правила фильтрации, учитывая множество параметров, что обеспечивает гибкость при настройке сетевых политик.

## ФИЛЬТРАЦИЯ СЕТЕВОГО ТРАФИКА С ПОМОЩЬЮ ВИРТУАЛЬНОЙ МАШИНЫ EBPF

eBPF [10] – это виртуальная машина, встроенная в ядро ОС Linux. Она позволяет запускать изолированные программы внутри ОС, что позволяет настраивать программы eBPF для добавления дополнительных возможностей к операционной системе в режиме реального времени. ОС при этом гарантирует безопасность и эффективность выполнения программ, как если бы они были скомпилированы нативно с помощью компилятора Just-In-Time (JIT). Существует множество проектов на основе eBPF, охватывающих широкий спектр сценариев использования, включая безопасность, мониторинг в реальном времени, а также сетевые технологии нового поколения [11].

Программы eBPF являются событийно-управляемыми и выполняются, когда ядро или приложение достигает определенной точки привязки (hook point). Предопределенные точки привязки включают системные вызовы, вход/выход из функций, трассировочные точки ядра, сетевые события и многие другие. Схема работы eBPF приведена на рис. 2.

В сетевой сфере eBPF позволяет пользователю загружать программы, которые выполняются каждый раз, когда пакет поступает на сетевой интерфейс или отправляется с него. Эти программы eBPF могут изменять, перенаправлять или отбрасывать пакеты [12].

Помимо этого, eBPF при реализации функций фильтрации сетевого трафика может использовать алгоритм LPM, который обрабатывает до 8 мегапакетов в секунду на одном ядре центрального процессора (ЦП), обработка пакета занимает O(1), что значительно ниже, чем у алгоритма RFC (recursive flow classification) [13].

## ФИЛЬТРАЦИЯ СЕТЕВОГО ТРАФИКА С ПОМОЩЬЮ IPSets

IPSets [14] – это фреймворк внутри ядра Linux, который может управляться с помощью утилиты ipset. В зависимости от типа, IPSets может хранить IP-адреса, сети, номера портов (TCP/UDP), MAC-адреса, имена интерфейсов или их комбинации таким образом, чтобы обеспечить высокую скорость сопоставления записи IPSets с набором полей заголовков трафика [15].

**Рисунок 2**
Принцип работы eBPF

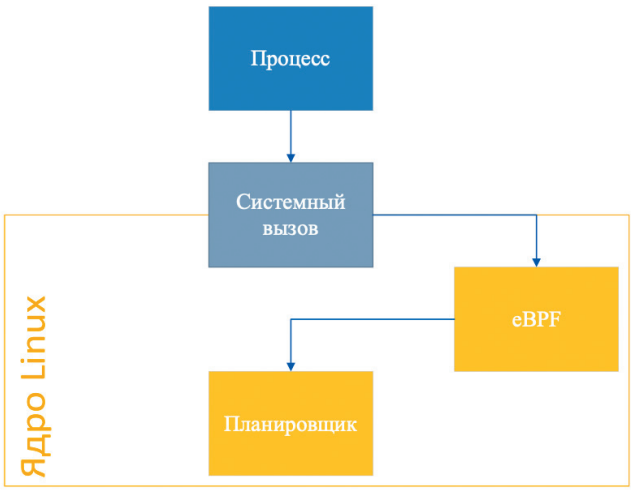
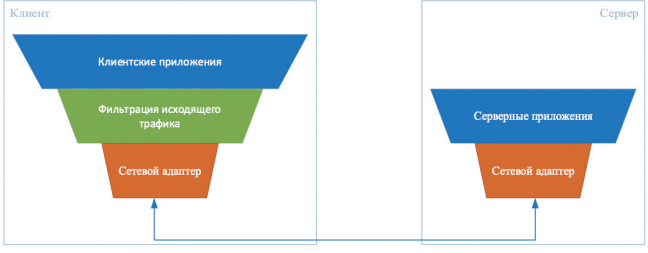



**Рисунок 3**
Схема экспериментального стенда



Ключевые особенности IPSets:

- хранение нескольких IP-адресов или номеров портов и сопоставление их с коллекцией с помощью iptables за один раз;
- динамическое обновление правила iptables для IP-адресов или портов без ущерба для производительности;
- работа со сложными наборами правил, основанными на IP-адресах и портах, и их выражение с помощью одного правила iptables, что позволяет значительно ускорить процесс создания правил.

## СРАВНИТЕЛЬНЫЙ АНАЛИЗ ФИЛЬТРАЦИИ СЕТЕВОГО ТРАФИКА НА ОСНОВЕ ЭКСПЕРИМЕНТАЛЬНОГО СТЕНДА

При проведении сравнительного анализа были рассмотрены следующие критерии эффективности фильтрации сетевого трафика:

- пропускная способность;
- загрузка ЦП;
- временная задержка на обработку правил.

Для проведения сравнительного анализа был разработан экспериментальный стенд, структура которого показана на рис. 3.

Для конфигурации iptables использовалась стандартная имплементация iptables и добавлялось правило DROP в цепочку OUTPUT. Для сопоставления правил iptables использует алгоритм линейного поиска (рис. 4). В процессе проведения тестов было принято решение использовать функцию iptables-restore, так как добавление каждого правила индивидуально занимало продолжительное время.

При конфигурации IPSets использовался сет типа hash:net (хэш:сеть). IP-адрес, который нужно проверить, хэшируется, и результат хэширования используется в качестве индекса в хэш-таблице. Каждый блок хэш-таблицы содержит массив сетей для этого хэша. Когда IP-адрес проверяется по набору IP-адресов типа hash:net, неизвестно, какой размер сети будет соответствовать. IPSets хэширует IP-адрес для всех возможных размеров сети. Например, если IP-адрес для сопоставления имеет вид 1.2.3.4, он ищет адреса в нотации CIDR 1.2.3.4/32, для 1.2.3.4/31 и так далее (т.е. осуществляет поиск конкретного адреса во всех возможных подсетях), пока не будут протестированы все 32 возможных размера сети. Принцип работы IPSets представлен на рис. 5.

Существует несколько вариантов конфигурации eBPF (рис. 6):

- CGROUP_SKB для фильтрации по группам управления (control group) на уровне сокета;
- SCHED_CLS и SCHED_ACT для фильтрации на уровне управления трафиком (traffic control);
- XDP для фильтрации на уровне сетевого интерфейса (network interface).

**Рисунок 4**
Алгоритм линейного поиска в iptables

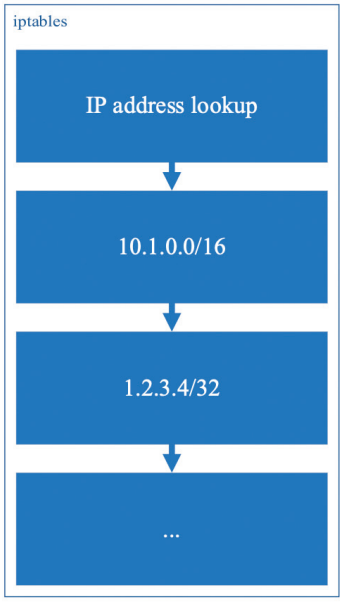



**Рисунок 5**
Конфигурация hash:net в IPSets

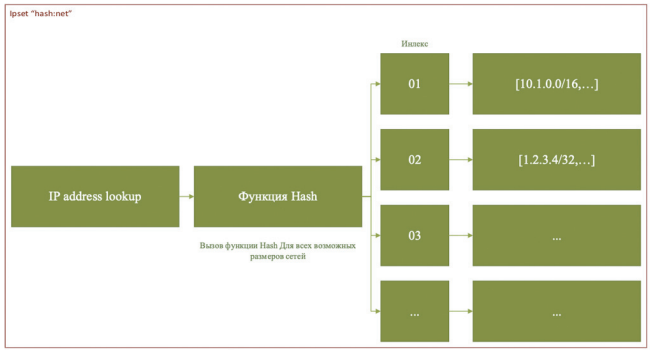



**Рисунок 6**
Реализация eBPF

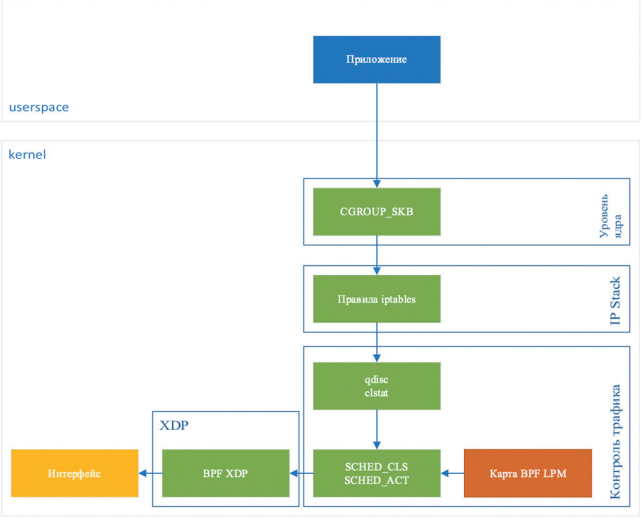


**Рисунок 7**
Результаты теста пропускной способности TCP

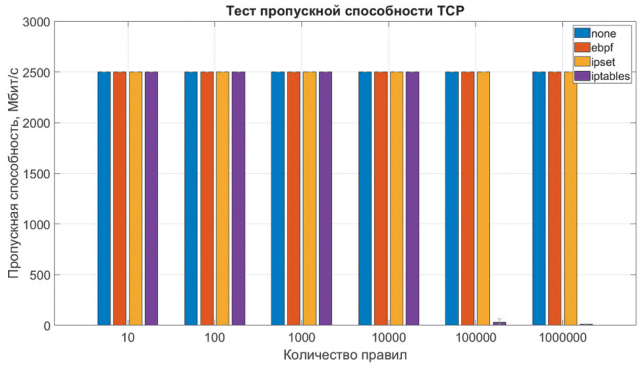



**Рисунок 8**
Результаты теста пропускной способности UDP

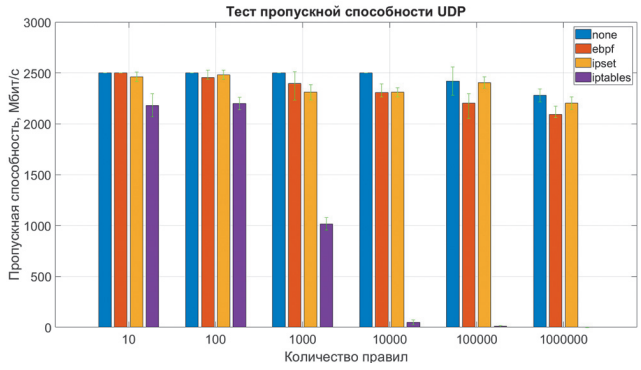



**Рисунок 9**
Результаты теста загрузки ЦП

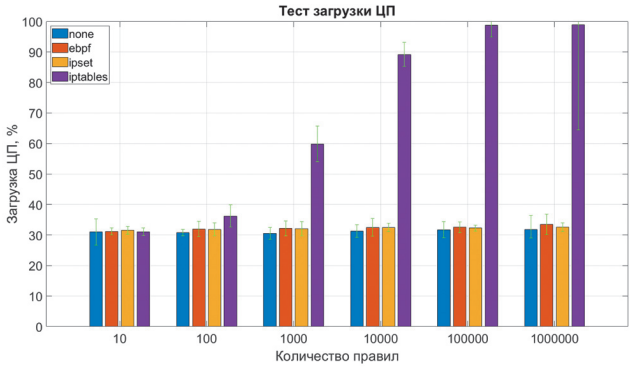
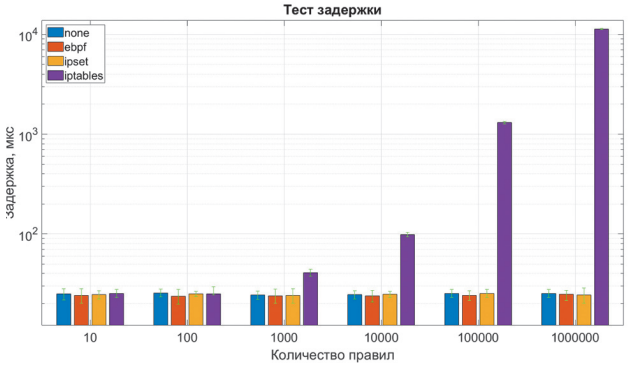



**Рисунок 10**
Результаты теста временной задержки




При экспериментальном тестировании необходимо применить одинаковую фильтрацию независимо от группы управления приложения, генерирующего трафик. Трафик, поступающий от приложений извне, или перенаправленный трафик, также должны подвергаться фильтрации. По этой причине фильтрация на уровне сокетов не подходит для данного случая.

Для измерения пропускной способности фильтрации трафика использовался генератор TCP и UDP трафика iperf3 [16]. Размер TCP-пакетов – 60 байт, UDP-пакетов – 1470 байт. В результате были получены диаграммы, представленные на рис. 7 и 8.

При проведении измерений загрузки ЦП генерировался трафик со скоростью 1 Гбит/с.

В результате была получена диаграмма, представленная на рис. 9.

На рис. 9 видно, что нагрузка на процессор при использовании утилиты iptables сильно увеличивается при применении более тысячи правил. Нагрузка на процессор при работе с фильтрами eBPF и IPSets остаётся практически одинаковой и почти постоянной, несмотря на увеличение числа правил.

Для измерения временной задержки использовалась стандартная Linux-утилита ping [17] с параметрами «-i 0.001» и «-c 1000», отправлявшая 1000 icmp пакетов с интервалом в 1 мс. В результате была получена диаграмма, представленная на рис. 10.

На диаграмме видно, что задержка не зависит от количества правил в фильтрах eBPF и IPSets, и составляет около 25–30 мкс для этих методов. С другой стороны, при использовании утилиты iptables задержка увеличивается линейно с ростом количества правил.

## ЗАКЛЮЧЕНИЕ

В данной статье проведен анализ способов фильтрации сетевого трафика в устройствах на базе ОС Linux. Для проведения сравнительного анализа результатов использования iptables, IPSets и алгоритма LPM [18] на базе eBPF был разработан экспериментальный стенд. Тесты пропускной способности, использования ЦП и временной задержки на проверку соответствия полей заголовков трафика и содержимого правил фильтрации показывают, что фильтры IPSets и eBPF хорошо масштабируются даже при наличии 1 млн правил. С другой стороны, программа iptables масштабируется хуже и даже при небольшом количестве правил (10 тыс. или даже при одной тысяче). Это ожидаемо, поскольку iptables использует линейный поиск в списке правил.

IPSets показал себя немного лучше, чем eBPF с картой LPM, демонстрируя немного более высокую пропускную способность в UDP-тесте и меньшее использование ЦП. Время поиска растет линейно с количеством добавленных различных значений префиксов.

## СПИСОК ЛИТЕРАТУРЫ


**1. Lee, J.** Federated learning-empowered mobile network management for 5G and beyond networks: From access to core / J. Lee, F. Solat, T.Y. Kim, H.V. Poor // IEEE Communications Surveys & Tutorials. – IEEE, 2024.

**2. Kumar, A.** Exploring the role of firewall technology in securing computer networks in the 5G environment / A. Kumar, S. Ahuja, G. Gupta // AIP Conference Proceedings. – 2024. – Vol. 3100. – Issue 1.

**3. Haseeb, R.H.U.** High-Speed Network DDoS Attack Detection: A Survey / R.H.U. Haseeb, A.H.M. Aman, M.K. Hasan et al. // Sensors. – 2023. – Vol. 23, Issue 15. – P. 6850.

**4. Muhammad, T.** A Optimizing Network Paths: In-Depth Analysis and Insights on Segment Routing / T. Muhammad, M.S. Kingsley, S. Ness // Journal of Data Acquisition and Processing. – 2023. – Vol. 38, Issue 4. – P. 1942-1963.

**5. Melkov, D.** Performance testing of Linux firewalls / D. Melkov, A. altis, . Paulikas // 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream). – IEEE, 2020. – P. 1-4.

**6. De Carvalho Bertoli, G.** Evaluation of netfilter and eBPF/XDP to filter TCP flag-based probing attacks / G. de Carvalho Bertoli, L.A. Pereire Junior, C.A.C. Marcondes, O. Saotome // XXII Symposium on Operational Applications in defense area (SIGE). – 2020. – P. 35-39.

**7. Likhar, P.** Impacts of Replace Venerable Iptables and Embrace Nftables in a new futuristic Linux firewall framework / P. Likhar, R.S. Yadav // 2021 5th International Conference on Computing Methodologies and Communication (ICCMC). – P. 1735-1742.

**8. Wahid, A.** The Implementation of Wireshark and IPtables Firewall Collaboration to Improve Traffic Security on Network Systems / A. Wahid, M.E. Firdaus, J.M. Parenreng // Internet of Things and Artificial Intelligence Journal. – 2021. – Vol. 1, № 4. – P. 249-264.

**9. Wu, M.H.** Mechanisms for Enhancing Network Services by Live Migration in the Network / M.H. Wu, C.Y. Chiu, J.Z. Wu // 2023 IEEE 6th Eurasian Conference on Educational Innovation (ECEI). – P. 10-13.

**10. Vieira, M.A.M.** Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications / M.A.M. Vieira, M.S. Castanho, R.D.G. Pacifico et al. // ACM Computing Surveys (CSUR). – 2020. – Vol. 53, Issue 1. – P. 1-36.

**11. Miano, S.** A framework for eBPF-based network functions in an era of microservices / S. Miano, F. Risso, M.V. Bernal et al. // IEEE Transactions on Network and Service Management. – 2021. – Vol. 18, Issue 1. – P. 133-151.

**12. Ghigoff, Y.** BMC: Accelerating memcached using safe in-kernel caching and pre-stack processing / Y. Ghigoff, J. Sopena, K. Lazri et al. // 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). – 2021. – P. 487-501.

**13. Антонова, В.М.** Система управления трафиком в перспективных мобильных сетях на основе технологий SDN/NFV / В.М. Антонова, И.Г. Бужин, Э.А. Гайфутдинов и др. // Информационные процессы. – 2023. – Т. 23, № 1. – С. 113-125.

**14. Khummanee, S.** FFF: Fast Firewall Framework to Enhance Rule Verifying over High-speed Networks / S. Khummanee, P. Songram, P. Pruksasri // ECTI Transactions on Computer and Information Technology (ECTI-CIT). – 2022. – Vol. 16, № 1. – P. 35-47.

**15. Loreti, P.** Implementation of accurate per-flow packet loss monitoring in segment routing over IPv6 networks / P. Loreti, A. Mayer, P. Lungaroni et al. // 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR). – IEEE, 2020. – P. 1-8.

**16. Fang, R.** Speeding up IPv4 connections via IPv6 infrastructure / R. Fang, G. Han, X. Wang et al. // SIGCOMM'21 Poster and Demo Sessions. – 2021. – P. 76-78.

**17. Chen, X.** Research on Linux Real-time and performance Evaluation for Loongson 3A3000 processor / X. Chen // Journal of Physics: Conference Series. – IOP Publishing, 2020. – Vol. 1453, Issue 1. – P. 012100.

**18. Scholz, D.** Key properties of programmable data plane targets / D. Scholz, H. Stubbe, S. Gallenmuller, G. Carle // 2020 32nd International Teletraffic Congress (ITC 32). – IEEE, 2020. – P. 114-122.


*Получено 25.07.24*