Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация сетевых подсистем мониторинга ядра ОС Linux»

Студент группы **ИУ7-56Б** _____ **В. М. Мансуров**
(Подпись, дата) (И.О. Фамилия)

Руководитель _____ **А. А. Оленев**
(Подпись, дата) (И.О. Фамилия)

2022 г.

РЕФЕРАТ

Научно-исследовательская работа 37 с., 8 рис., 3 табл., 17 ист., 1 прил.

ЯДРО LINUX, СЕТЕВАЯ ПОДСИСТЕМА, СЕТЕВОЙ МОНИТОРИНГ, МОДИФИКАЦИЯ КОДА ЯДРА, ЗОНДИРОВАНИЕ ЯДРА, ТОЧКИ ТРАССИРОВКИ, EBPF, СЕТЕВОЙ СТЕК, FTRACE, KPROBE

Объект исследования — сетевая подсистема ядра Linux.

Цель работы — Классификация методов сетевого мониторинга ядра Linux, выбор методов, которые наилучшим способом решают необходимые задачи.

Поставленная цель достигается путем рассмотрения и классификации существующих и применяемых методов сетевого мониторинга ядра Linux.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

Application Programming Interface — «Интерфейс прикладного программирования» - набор инструментов программирования, который разрешает программе взаимодействовать с другой программой или операционной системой и помогает разработчикам программного обеспечения создавать свои собственные приложения [1].

Протокол — формализованные правила, определяющие последовательность и формат сообщений, которыми обмениваются сетевые компоненты, лежащие на одном уровне модели сетевого взаимодействия в разных узлах. [9].

IP-пакет или пакет — отформатированная единица данных, переносимая сетью с коммутацией пакетов. [9].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В текущей расчетно-пояснительной записке применяется следующие сокращения и обозначения.

ОС — Операционная система.

IP — Internet Protocol.

OSI — Open Systems Interconnection.

TCP — Transmission Control Protocol.

IETF — Internet Engineering Task Force.

BSD — Berkeley Software Distribution.

API — Application Programming Interface.

IPX — Internetwork Packet Exchange.

LAN — Local Area Network.

BPF — Berkeley Packet Filters.

eBPF — Extended Berkeley Packet Filter.

ВВЕДЕНИЕ

Организация взаимодействия между устройствами и программами в сети сложная задача. Сеть объединяет разные оборудование, операционные системы и программы — это было бы невозможно без принятия общепринятых правил, стандартов. В области компьютерных сетей существует множество международных и промышленных стандартов, среди которых следует особенно выделить международный стандарт OSI и набор стандартов IETF.

В ОС такую задачу реализуют сетевая подсистема, что позволяет иметь широкий спектр сетевых возможностей. Сетевая подсистема, выполняющаяся в режиме ядра, отвечает за управление сетевыми устройствами ввода-вывода, но кроме этого на нее также возложены задачи маршрутизации и транспортировки пересылаемых данных. Современные ОС Linux требует контроля по причине того, что безопасность ядра не идеальна в том числе и сетевая подсистема ядра [2].

Для отладки сетевых ошибок проверяются все узлы, участвующие в сетевом взаимодействии: отправляющий, связующие и принимающий. Однако из-за сложной конфигурации, в сети возникают не очевидные связи и взаимодействий между различными элементами, что сильно осложняет процесс поиска источника неполадки даже в одном узле. Даже возникает ситуация, в которой непонятно, с какой стороны подступиться проблеме. Тогда разработчику придется перебирать возможные причины возникновения ошибки, используя набор доступных для сетевой отладки инструментов и полагаясь на профессиональный опыт разработчика. Такой бессистемный подход приводит к трудностям устранения сбоев .

Целью работы является провести анализ существующих средств мониторинга сетевой подсистемы ядра ОС Linux.

Для достижения поставленной цели необходимо решить следующие задачи:

— провести анализ предметной области сетевой подсистемы ядра ОС Linux;
— провести обзор существующих подсистем и средств сетевого мониторинга ядра ОС Linux;
— сформулировать критерии сравнения средств сетевого мониторинга ядра;
— классифицировать существующие подсистемы и средства сетевого мониторинга.

1 Анализ предметной области

1.1 Ядро Linux

Ядро Linux — основной внутренний компонент ОС Linux, отвечающие за поддержку работу с накопителями, управление работы процессора посредством переключения между выполняемыми задачами, распределение системными ресурсами, управление аппаратным обеспечением и обеспечение взаимодействие приложения с аппаратным обеспечением, а также ядро принимает сообщения и пакеты данных из сети и отправляет их в сеть [3].

Прежде всего для ядра ставятся задачи обеспечение среды выполнения для программ в ОС, взаимодействие с аппаратными компонентами и обслуживание их низкоуровневые элементы. Кроме того в настоящее время в функции ядра включают управление процессорами графической оболочки, обеспечивающей интерфейс между пользователем и ОС.

Ядро Linux является монолитным с модульной конструкцией, которое реализовано в виде одного процесса, выполняющий в одном адресном пространстве. Такое ядро обычно хранится на диске в виде одного статического бинарного файла. Все службы ядра находятся и выполняются в одном большом адресном пространстве ядра. Взаимодействия в ядре осуществляются очень просто, потому что все, что выполняется в режиме ядра, выполняется в одном адресном пространстве. В отличие от микроядра, который разделяет службы ядра на несколько процессов, называемыми серверами. Ядро может вызывать функции непосредственно, как это делают пользовательские приложения.

1.2 Подсистемы ядра

Ядро Linux разделяется на ряд подсистем, показанные на рисунке 1, как на высоком, так и на низких уровнях. Такое разделение позволяет упростить разработку ядра и сделать его более гибким. Каждая подсистема реализует свой функционал, такие как: управление памятью, управление и взаимодействие процессов, файловая система, сетевые стек, которое используется другими подсистемами. Компоненты системы разнесены по слоям, несмотря на то, что модели или подсистемы работают в «одном слое».

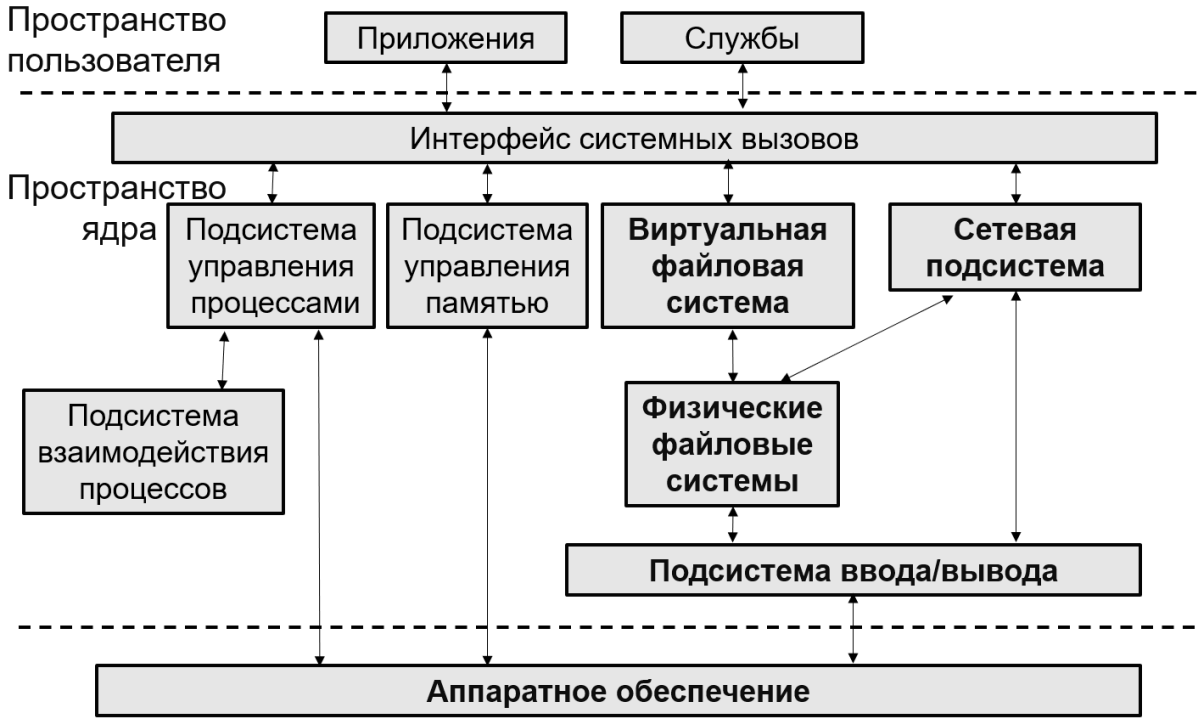

Рисунок 1 – Общая архитектура Linux

В ОС Linux три главных уровня, показанные на рисунке 1. В основе расположены аппаратные средства, включающие память, процессор и выполняющие вычисления, запросы на чтение из памяти и запись в нее, а также устройства такие как жесткий диск и сетевые интерфейсы. Уровнем выше расположено пространство ядро (от англ. kernel space), включающее системные переменные и область памяти ядра. Выше же пользовательские процес-

сы, которыми управляет ядро, называемым пространством пользователя (от англ. user space).

В Linux выделяют два режима работы ее программных средств — режим ядра (от англ. kernel mode) или привилегированным режимом, и пользовательский режим (от англ. user mode). Основное различие двух режимов состоит в привилегиях доступа к аппаратным средствам — оперативной памяти, процессору и устройства ввода-вывода, к которым разрешен полный доступ из режима ядра и ограниченный доступ из пользовательского режима. Это сильное преимущество, но может быть опасным, поскольку позволяет процессам ядра с легкостью нарушить работу всей системы. В режиме пользователя, для сравнения, доступен ограниченный объем памяти и разрешены безопасные инструкции для процессора. Пространством пользователя называют участки оперативной памяти, которые могут быть доступны пользовательским процессам. Если какой-либо процесс завершается с ошибкой, ее последствия будут ограниченными и ядро сможет их очистить. Это означает, что, если, например, произойдет сбой в работе браузера, выполнение научных расчетов, которые вы запустили на несколько дней в фоновом режиме, не будет нарушено.

1.3 Сетевая подсистема ядра

Сетевая подсистема ядра Linux [4, 5] обеспечивает взаимодействие процессов, выполняющихся на разных узлах сети, то есть дает возможность сетевого взаимодействия между приложениями.

Сетевая подсистема ядра Linux реализует следующую функциональность:

— поддержку взаимодействия процессов с помощью механизмов сокетов (sockets);

— реализацию стеков сетевых протоколов (TCP/IP, UDP/IP, IPX/SPX и другие);
— поддержку сетевых интерфейсов или драйверов;
— обеспечение маршрутизации пакетов (routing);
— обеспечение фильтрации пакетов (netfilter).

1.3.1 Состав подсистемы

В состав сетевой подсистемы входят модули [6], показанные на рисунке 2.

— Драйверы сетевых устройств обеспечивают связи с помощью аппаратных средств. Для каждого сетевого устройства существует свой драйвер.
— Модуль аппаратно-независимого интерфейса обеспечивает пользовательскому процессу единый интерфейс ко всем физическим сетевым устройствам.
— Модуль сетевых протоколов предназначен для реализации всех возможных транспортных протоколов в сети.
— Модуль интерфейсов независимых от протоколов обеспечивает интерфейсы, которые не зависят от протоколов и физических устройств. Этот модуль использует ядро для доступа к сети без привязки к протоколами и физическим устройствам.
— Системный интерфейс нужен для взаимодействия ядра или пользователя с сетевой подсистемой.

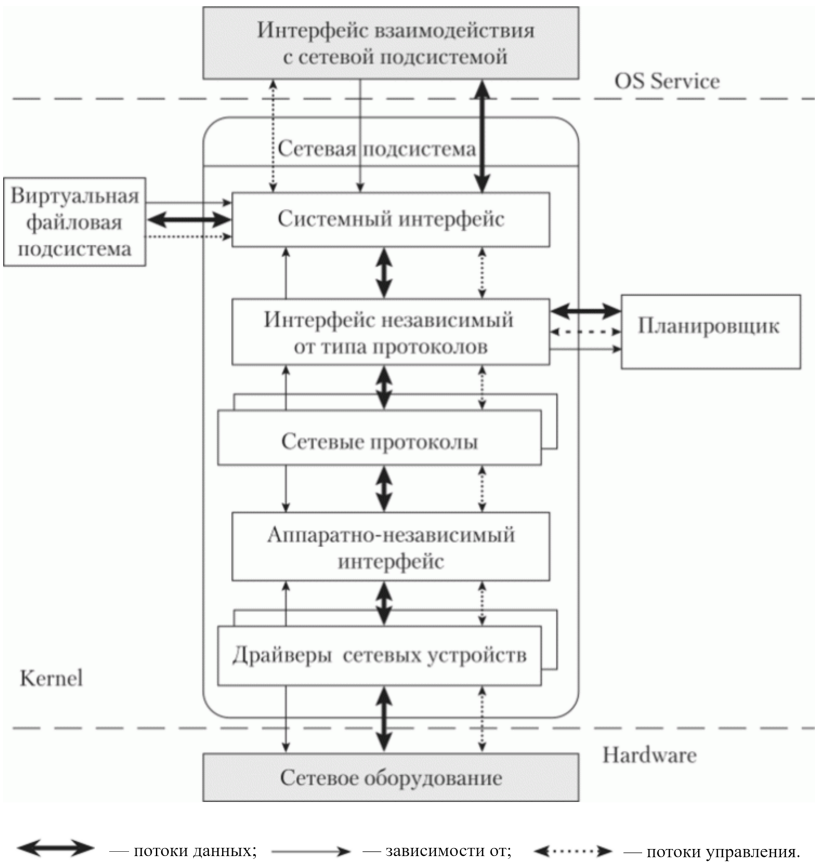

Рисунок 2 – Архитектура сетевой подсистемы ядра Linux

1.3.2 Зависимости сетевой подсистемы

Сетевая подсистема обращается к планировщику для приостановления и возобновления работы процессов, пока они находятся в ожидании завершения запроса оборудования, что приводит к зависимости других подсистем. И применяет планировщик для временной синхронизации функций обмена данными. Также поддерживает VFS посредством реализации элементов логической файловой системы в части NFS, что приводит к зависимости VFS от сетевой подсистемы и контролю потоков данных между подсистемами. Сетевая подсистема зависит от IPC, путем использования демона kerneld динамической загрузки и выгрузки исполняемых модулей.

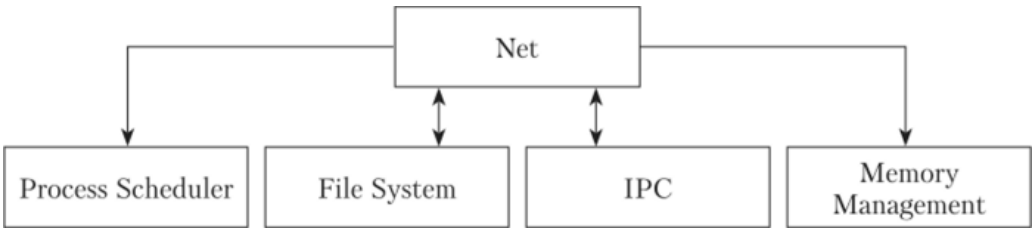

Рисунок 3 – Зависимости сетевой подсистемы

1.3.3 Сетевой стек

Каждый объект в сети, с точки зрения Linux — сокеты, представляющая собой структуру данных в сетевом узле сети, которая служит конечной точкой для отправки и приема данных по сети. Сетевая подсистема основана на использовании модели сокетов, введенных в ОС BSD, и поддерживает множество различных сетевых стеков протоколов. Данные многоуровневые протоколы применяются для решения задач сетевых взаимодействий, заключающийся в разбиении процесса коммуникации на набор уровней с четко определенными способами взаимодействия уровней на одном узле и на соседних узлах для обеспечения коммуникации различного оборудования. Реализация сетевой поддержки в ОС-ах основана на структуре сокетов struct socket, содержащая поле идентификатора типа сокета (потоковый или дейтаграммный), состояние сокета (в состоянии соединения или нет), поле с флагами (модифицируют работу сокета), указатель на структуру со списком операций для выполнения сокетом и другие данные. На листинге 1 показана полная структура сокета.

Сетевая реализация построена так, чтобы не зависеть от конкретики протоколов. Таким образом в подсистеме существует сетевой стек, который оснащен набором интерфейсов, которые варьируются от протоколонезависимых (protocol agnostic), таких как интерфейс уровня сокетов или уровня устройств, до специальных интерфейсов конкретных сетевых протоколов.

Листинг 1 – Структура сокета struct socket [7]

```
1 struct socket {
2     socket_state state;
3     short type;
4     unsigned long flags;
5     struct socket_wq __rcu * wq;
6     struct file * file;
7     struct sock * sk;
8     const struct proto_ops * ops;
9 };
```

Уровень сокетов представляет собой стандартный API к различным сетевым протоколам для сетевой подсистемы, обеспечивающий способ управления соединениями и передачи данных между конечными точками, от доступа к данных и блокам данных протокола IP/PDU, и до протоколов TCP/UDP.

В то время как работа в сети отсылается к модели сетевого взаимодействия по OSI, сетевой стек в Linux использует модель TCP/IP [8, 9], включающая в себя 4 уровня:

1) уровень сетевых интерфейсов (канальный уровень) относится к драйверам устройств, обеспечивающим доступ к физическому уровню, который может состоять из многочисленных сред, таких как последовательные каналы или устройства Ethernet, описываются как сетевые интерфейсы;
2) уровень межсетевого взаимодействия (сетевой уровень) обеспечивает работу базовой службы доставки пакетов по назначению;
3) транспортный уровень обеспечивает надежную доставку данных со сквозным обнаружением и устранением ошибок;
4) прикладной уровень отвечает за взаимодействие с приложениями и процессами на хостах, также определяются пользовательские интерфейсы процесса или приложения, наблюдается работа протоколов и служб — FTP, Telnet и другие.

Несмотря на обилие возможностей сетевая подсистема ориентирована на обслуживание протоколов Ethernet и TCP/IP, которые условно вписывается в модель OSI, включающая в себя 7 уровней, показанные на рисунке 4.

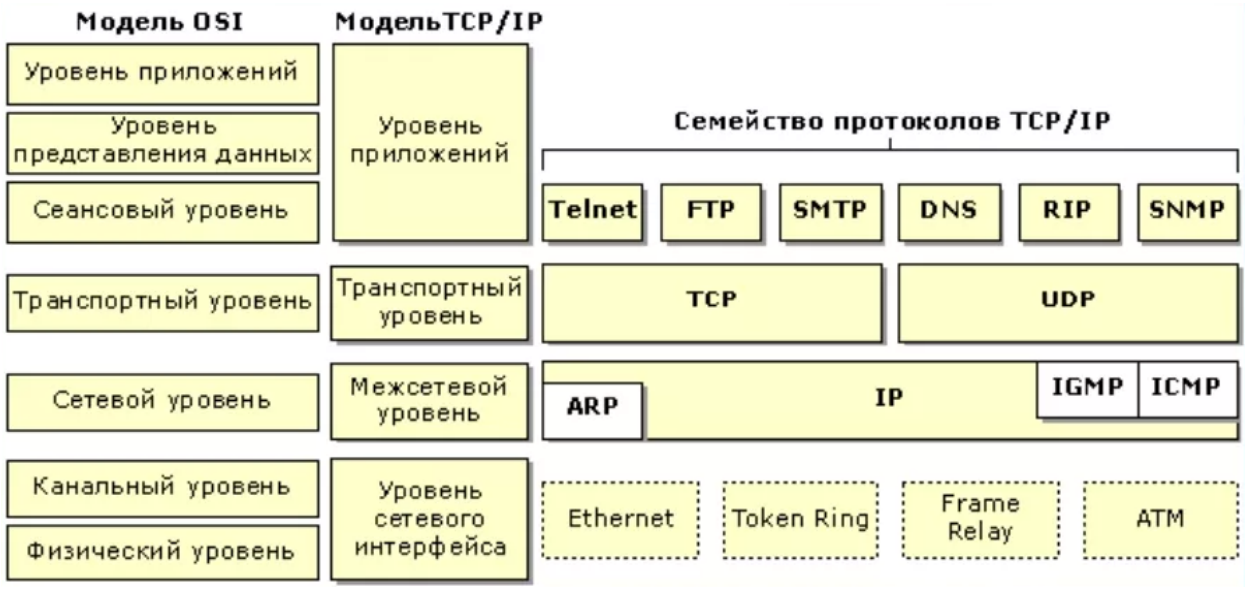

Рисунок 4 – Модель OSI и TCP/IP

1.3.4 Сетевые интерфейсы

Сетевые интерфейсы создаются поддерживающими их драйверами — модулями ядра Linux. В общем случае, разработчик драйвера (модуля ядра) специфического сетевого устройства может выбрать имя для его интерфейса произвольно (определяется драйвером)

Основной структурой данных описывающей сетевой интерфейс — struct net_device и представленная на листинге 2, описанная в <linux/netdevice.h>. Она содержит не только описание аппаратных средств, но и конфигурационные параметры сетевого интерфейса по отношению к протоколам. Данная структура часто меняется от версии к версии ядра.

Листинг 2 – Основная структура данных

```
1  struct net_device {
2      char  name[ IFNAMSIZ ] ;
3      ...
4      unsigned long  mem_end;    /* shared mem end      */
5      unsigned long  mem_start; /* shared mem start    */
6      unsigned long  base_addr; /* device I/O address */
7      unsigned int   irq;        /* device IRQ number   */
8      ...
9      unsigned       mtu;        /* interface MTU value      */
10     unsigned short type;       /* interface hardware type */
11     ...
12     /* Interface address info. */
13     unsigned char  perm_addr[ MAX_ADDR_LEN ]; /* permanent hw
           address     */
14     unsigned char  addr_len;                  /* hardware address
           length */
15     ...
16 }
```

А вот основной структурой управления передачей пакетов и экземпляров данных между сетевыми уровнями, построена работа всей подсистемы — буферы сокетов struct sk_buff, определенная в <linux/skbuff.h> и представленная на листинге 3. Буфер сокетов состоит их двух частей: данные управления struct sk_buff и данные пакета, указываемые в struct sk_buff указателями head и data. [4, 6]. В этой структуре объединены все типы пакетов, используемых в сетевой подсистеме, члены которой включают:

— счетчики запросов сокета на чтение и запись в память;

— флаги, определяющие поведение сокета;

— поля управления в буфере;

— последовательности действий, предписанным протоколом TCP/IP;

— очередь ожидания (struct sk_queue_head) для блокирования операций

чтения и записи, посредством полей next и prev.

Листинг 3 – Структура данных управления передачей пакетов

```
struct sk_buff {
    struct sk_buff *next;
    struct sk_buff *prev;
    ...
    __u16           inner_transport_header;
    __u16           inner_network_header;
    __u16           inner_mac_header;
    __be16          protocol;
    __u16           transport_header;
    __u16           network_header;
    __u16           mac_header;
    ...
    sk_buff_data_t      tail;
    sk_buff_data_t      end;
    unsigned char       *head, *data;
    unsigned int        truesize;
    refcount_t          users;
    ...
};
```

Задача сетевого интерфейса — быть тем местом, в котором:

— создаются экземпляры структуры struct sk_buff, по каждому принятому из интерфейса пакету (здесь нужно принимать во внимание возможность сегментации IP пакетов), далее созданный экземпляр структуры продвигается по стеку протоколов вверх, до получателя пользовательского пространства, где он и уничтожается;

— исходящие экземпляры структуры struct sk_buff, порождённые на верхних уровнях протоколов пользовательского пространства отправляются, а сами экземпляры структуры после этого — уничтожаться. Более детально эти вопросы рассмотрены, при обсуждении прохождения па-

кетов сквозь стек сетевых протоколов.

1.3.5 Путь пакета данных сквозь стек протоколов

Структура вложенности заголовков сетевых уровней в точности соответствует структуре инкапсуляции сетевых протоколов протоколов внутри друг друга, это позволяет обрабатывающему слою получать доступ к информации, относящейся только к нужному ему слою. Экземпляры данных типа struct sk_buff во-первых возникают при поступлении очередного сетевого пакета из внешней физической среды распространения данных. Об этом событии извещает прерывание (IRQ), генерируемое сетевым адаптером. При этом создается или извлекается из пула источников экземпляр буфера сокета, заполняется данными из поступившего пакета. Все эти действия выполняются не в самом обработчике верхней половины прерываний от сетевого адаптера, а в обработчике отложенного прерывания NET_RX_SOFTIRQ. Затем этот экземпляр дальше вверх от сетевого слоя к слою, до приложения прикладного уровня, которое является получателем пакета. На этом экземпляр данных буфера сокета уничтожается. Во-вторых возникают в среде приложения прикладного уровня, которое является отправителем пакета данных. Пакет помещается в созданный буфер сокета, который начинает перемещаться вниз от сетевого слоя к слою, до достижения канального уровня, где осуществляется физическая передача данных пакета через сетевой адаптер в среду распространения. В случае успешного завершения передачи буфер сокета уничтожается. При отсутствии подтверждения отправки (IRQ) обычно делается несколько повторных попыток, прежде, чем принять решение об ошибке канала.

1.4 Сетевой мониторинг ядра

Сетевой мониторинг ядра Linux — отслеживание пути пакетов и выявление наличие ошибок сетевых интерфейсов сетевой подсистемы, что относится к вмешательству в работу сетевой подсистемы. С одной стороны большинство проблем с сетью возникают как раз на физическом и канальном уровнях, но с другой стороны приложения, работающие с сетью оперируют на уровне TCP сессий и не видят, что происходит на более низких уровнях. Поэтому путем использования сетевого мониторинга можно выявить:

— проблемы производительности диска (хранилища);

— проблемы производительности памяти и процессора;

— узкие места в сети.

Сетевой стек устроен сложно, и не существует универсального решения на все случаи жизни. Если критически важны производительность и корректность при работе с сетью, то придется потратить немало времени, сил и средств в то, чтобы понять, как взаимодействуют друг с другом различные части системы.

В идеале следует измерять потери пакетов на каждом уровне сетевого стека. В этом случае необходимо выбрать, какие компоненты нуждаются в настройке. Именно на этом моменте, как мне кажется, сдаются многие. В ряде случаев, вероятно, система бывает настолько пронизана взаимосвязями и наполнена нюансами, что если вы пожелаете реализовать полезный мониторинг или выполнить настройку, то придется разобраться с функционированием системы на низком уровне. В противном случае просто используйте настройки по умолчанию. Этого может быть достаточно до тех пор, пока не понадобится дальнейшая оптимизация и вложения для отслеживания этих настроек.

2 Существующие решения

В данном разделе рассмотрены основные подсистемы и средства сетевого мониторинга ядра Linux, которые используются в настоящее время.

2.1 Обзор методов сетевого мониторинга ядра Linux

2.1.1 Утилиты для сетевого мониторинга

Для современных Linux существует множество утилит для конфигурации и устранения неполадок сетевой подсистемы ядра. В основном это инструменты командой строки. Наиболее распространенные и часто используемые из них iproute2, ethtool, ping, traceroute, nslookup, netcat, iptables, tcpdump [8], с помощью которых можно узнать конфигурацию сетевых пакетов, проверить наличие соединения и работоспособность Domain Name System (DNS), просмотреть таблицу маршрутизации, изучить содержание сетевых пакетов и многое другое, что может помочь определить сбои в сетевой подсистеме.

Путем использования данных утилит с одной стороны обеспечивает безопасность и производительность, но с другой — такое множество средств заточено под определенную задачи и их функциональность ограничена программным интерфейсом ядра Linux. Многие инструменты имеют документацию, благодаря которой можно разобраться для чего необходимо и как использовать утилиту. Все же решая какую-либо сложную задачу, а многие задачи мониторинга сетевой подсистемы и есть такие, поэтому необходимо использовать комбинацию данных утилит, если такие утилиты имеются.

2.1.2 Модификация кода ядра Linux

Открытость ОС Linux позволяет реализовать средства получения информации о событиях, происходящих в сетевой подсистеме ядра, путем модификации исходного кода ядра, что дает возможность находить сбои.

Модификация ядра Linux — изменение или добавление кода, которое не несет изменение структур ядра Linux. Модификация с целью сетевого мониторинга изменение кода сетевой подсистемы ядра Linux, путем добавление функционала вывода информации в системный журнал. [10]

Для отслеживания пути пакетов необходимо разместить код, перехватывающий события «ловушки» и выводимые информацию либо в системный журнал, либо в пользовательское приложение. На рисунке 5 показан пример структуры добавление кода в ядро Linux для отслеживания пути пакета, где модуль регистрации событий — динамически загружаемый модуль ядра Linux. Каждая ловушка сигнализирует о конкретном событии с конкретным пакетом, поэтому в обработчик перехваченных событий передается идентификатор пакета, место возникновения события, код события и дополнительная параметры. В качестве идентификатора пакета используется структура sk_buff, в которой содержатся пакет. Ловушки размещаются в начале и в конце сетевых функции фильтрации обрабатываемых пакетов в случае удовлетворению фильтра вывод необходимой информации. Благодаря чему можно определить как изменился пакет при возникновении ошибки и к чему это привело. [11]

Модификации ядра имеют доступ ко всему функционалу, включая системные вызовы. При неверном изменении кода ядра возникают критические ошибки, которые могут привести к нарушению работоспособности системы, вследствие чего нельзя достичь необходимого уровня безопасности. Кроме того для добавления кода в ядро необходимо обладать профессиональными

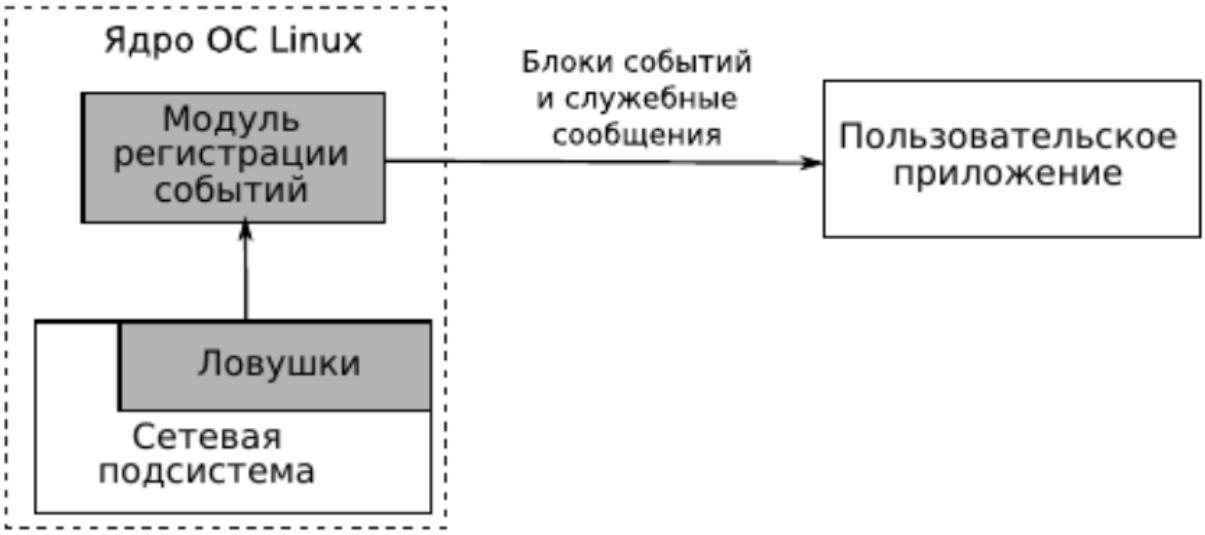

Рисунок 5 – Структурная схема системы слежения пути пакета

навыками работы с языком C и компилятором с инструментами конфигурации ядра. Модификация и пересборка ядра делают невозможный запуск на реальной системе, что делает процесс длительным и осложненным при достижении низких расходных ресурсов при фильтрации пакета в каждой функции ядра. Но с другой стороны модификация кода дает возможность решить любую задачу связанную с мониторингом пакета в сетевой сети.

2.1.3 Зондирование ядра Linux

Второй способ сетевого мониторинга ядра Linux, основывается на встраивании модулей ядра, берет свое начало аналоговой реализации от IBM — DProbes. DProbes включали в себя, помимо основного зондирующего механизма, обратный интерпретатор, обработчики зондов которого могут быть реализованы как простые функции на языке C, которые будут выполняться в контексте ядра, если они скомпилированы как модуль ядра или даже скомпилированы в ядро. [12]

Использование зондирование ядра (от англ. kernel probe, kprobes) [13] позволяет входить в работающее ядро для целей отладки, трассировки, оцен-

ки производительности, нахождение ошибок и т.п., путем установления точек останова. По достижению точки останова, возникает ловушка, регистры сохраняются, а управление передается к соответствующей функции написанного модуля ядра. После завершения данной функции работа ядра возвращается.

Большая часть функционала ядра поддерживает зондирование, поэтому с его помощью можно исследовать все ядро. Для того, чтобы отследить путь сетевого пакета с помощью зондирования ядра добавляются пользовательские модули ядра для всех функций, участвующих в обработке пакета. В каждой модуле обозначается имя соответствующей функции и описывается метод, который будет фильтровать обрабатываемый пакеты.

При использовании зондировании ядра безопасность намного лучше по сравнению с модификацией ядра благодаря вынесению функциональности в отдельные модули. Однако вероятность нарушить работу ядра остается высоким по причине добавления кода и при исследовании кода ядра могут требуется осторожность, так как могут меняться набор регистров, включая указатель команд. С одной стороны, за счет динамической загрузки модулей ядра появляется возможность запуска на работающей системе, но осложняется путем пересборки модулей при изменении фильтра. Использование ресурсов становится больше за счет проведения дополнительных операций при передаче управления в описанные модули. С другой стороны, реализация независима от конкретной сборки ядра упрощается за счет того, что для разных версий нужно адаптировать только имена функций.

2.1.4 Точки трассировки

Точки трассировки (от англ. tracepoints) — статически определенные места в коде ядра Linux, в которых можно запускать пользовательский код

[14, 15]. Точки трассировки, размещенная в коде, обеспечивает перехватчик для вызова функции (пробы), указанные во время выполнения.

Точка трассировки может находится в двух состояниях: «включена» (к ней подключен зонд) или «выключена» (зонд не подключен). Когда точка трассировки «выключена», она не имеет никакого эффекта, за исключением времени проверки условия для перехода и пространство для вызова функции и данных. Когда точка трассировки «включена», предоставляемая функция вызывается каждый раз при выполнении точки трассировки в контексте выполнения вызывающего объекта. После завершения предоставленной функции, выполнение ядра возвращается в нормальный вид.

Использование данного метода схоже с использованием зондирование ядра, то есть для всех наблюдаемых точек трассировки создаются модули ядра с реализаций фильтрации сетевых пакетов. Загружаемые модули ядра могут выполнять написанный код при попадании в точку трассировки, что делает их очень эффективными. Также точки трассировки определяются в коде, а не привязываются к имени функции, что позволяет решить проблему с зависимостью от версии ядра. Важным параметром точек трассировок является принадлежность к стабильному API Linux, вследствие если точка объявлена ее нельзя убрать или переместить. Поэтому разработчики ядра уже добавили многие точки трассировки для безопасности подсистем, но количество данных точек должно быть минимально, что приводит к меньшим расходам ресурсов.

2.1.5 Function Trace

Function Trace (ftrace) был разработан Стивенном Ростедтом и добавлен в ядро в 2008 году. Ftrace внутренний механизм трассировки, позволяющий получить различную информацию о вызовах функции ядра Linux. Он исполь-

зуется для отладки или анализа задержек, определения проблем с производительностью, происходящие за пределами пользовательского пространства, отслеживать контекстные переключения, измерять время обработки прерываний. Хотя ftrace считается механизмом трассировки, он на самом деле является основой нескольких различных утилит трассировки [16].

Принцип работы ftrace, которые иллюстрируется на рисунке 6, заключается в выполнении кода для трассировки в выполнении кода для трассировки в вызовах функции ядра Linux, которые ставятся в начало исполнения всех функций яда Linux, так как при сборке выставляются флаги «-pg» компилятора «gcc».

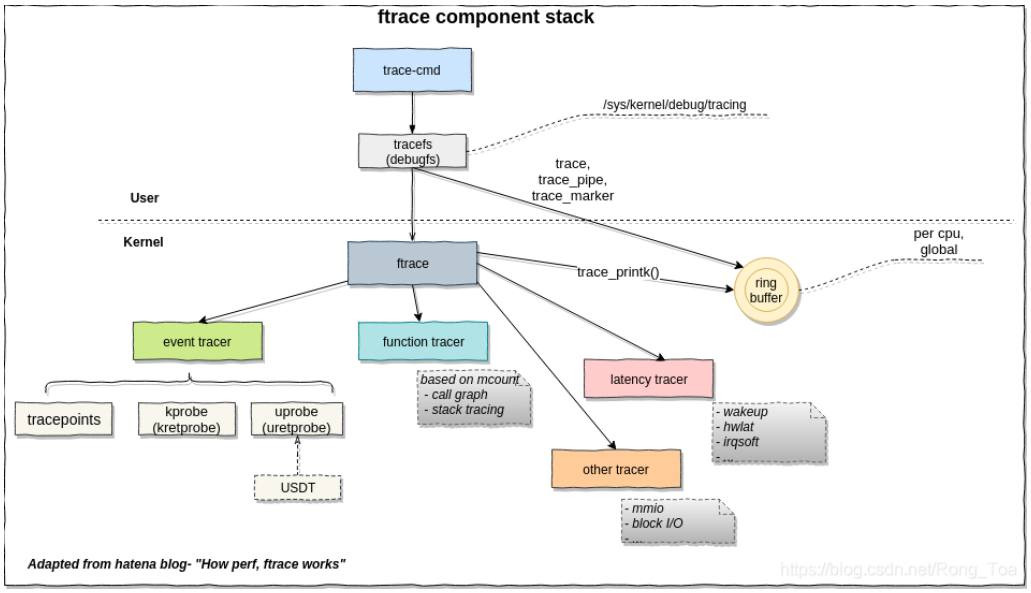

Рисунок 6 – Принцип работы механизма трассировки ftrace

Ftrace использует файловую систему tracefs, которая ориентирована на подсистему трассировки и получение доступа к интерфейсу трассировки через каталог без необходимости монтировать debugfs. При этом создается каталог трассировки «/sys/kernel/debug/tracing». Также применяется в ситуациях, когда использования debugfs невозможно из соображения безопасности, т.е. подсистемы ядра могут выводить через debugfs закрытые сведения.

Для мониторинга сетевой подсистемы ядра Linux с помощью ftrace используется следующий функционал:

— function — трассировщик вызовов функций ядра без возможности получения аргументов;
— function_graph — трассировщик вызовов функций ядра как function, который показывает все вызовы из определенных фильтров функции, но указывает точку входа и выхода с помощью чего можно отслеживать функции с подвызовами и измерять время выполнения для каждой функции;
— blk — трассировщик вызовов и событий ядра, связанных с вводом-выводом на блочные устройства, а именно использованием утилиты blktrace;
— mmiotrace — трассировщик операций ввода-вывода.

Для отслеживания действий сетевой подсистеме лучше использовать трассировщик function_graph, фильтре которого указывается имя функции, которые участвую в отслеживаемых процессах сетевой подсистемы.

Использование ftrace не требует модификации кода или добавления дополнительных модулей ядра Linux, поэтому интерфейс является неизменяемым, то есть не зависит от сборки ядра Linux. Самой же противоречивой особенностью ftrace является невозможность добавления пользовательского кода в ядро Linux при трассировке. С одной стороны, благодаря этому повышается безопасность и понижаются расходы ресурсов, но с другой — ухудшается гибкости под определенную задачу.

2.1.6 Extended Berkeley Packet Filter

В 1992 году Стивен Маккейн и Ван Якобсон опубликовали новую архитектуру для захвата пакетов на уровне пользователя, где описали способ реализации фильтра сетевых пакетов для ядра Unix, который работал в 20 раз быстрее, чем все остальные методы фильтрации пакетов.

Berkeley Packet Filter (BPF) — технология, которая позволяет добавлять программы или модули в ядро Linux без изменения исходного кода ядра [17]. BPF представил два серьезных нововведения в области фильтрации пакетов:

— новую виртуальную машину (ВМ), предназначенную для эффективной работы с центральным процессором на основе регистров;

— возможность использования буферов для каждого приложения способных фильтровать пакеты без копирования всей информации о них.

Extended Berkeley Packet Filter (eBPF) расширение реализации BPF, разработанной в 2014 году Алексей Старовойтов. Новый подход был оптимизирован для современного оборудования, благодаря чему результирующий набор команд работает быстрее, чем машинный код, сгенерированный старым интерпретатором BPF. Расширенная версия, показанные на рисунке 7, также увеличила число регистров в виртуальной машине BPF с двух 32-битных регистров до десяти 64-битных. Увеличение количества регистров и их глубины позволило писать более сложные программы, поскольку разработчики могли свободно обмениваться дополнительной информацией, используя параметры функций. Эти изменения наряду с прочими улучшениями привели к тому, что расширенная версия BPF стала в четыре раза быстрее оригинальной реализации BPF.

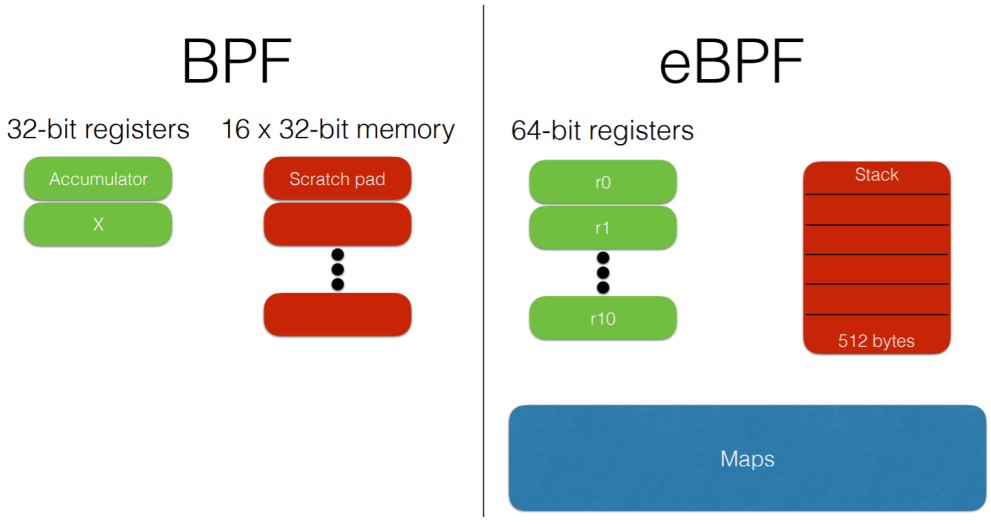

Рисунок 7 – Отличие между BPF и eBPF технологий

eBPF стал подсистемой ядра верхнего уровня в современных ОС Linux и больше походить на модули ядра с сильным акцентом на безопасность, чтобы предотвратить системные сбои и вредоносное поведение каких-либо программ, и стабильность, по причине того что не требует перекомпиляции.

eBPF работает по следующему принципу, показанному на рисунке 8. В своей основе eBPF использует привилегированную способность ядра видеть и контролировать все ресурсы системы, а компиляторы подобные GNU Compiler Collection (GCC) обеспечивает поддержку BPF, что позволяет скомпилировать код на языке C в байт-код. С помощью eBPF запускаются изолированные программы в привилегированном контексте, которые используются для доступа к оборудованию и службам из области ядра Linux. eBPF позволяет пользовательским приложениям упаковать логику, выполняемую в ядре при возникновении событий. После компиляции кода приложения использует «верификатор», чтобы убедиться в безопасности для запуска в пространстве ядра. Если код прошел все проверки успешно, программа BPF будет загружена в ядро и преобразована из байт-кода BPF в машинный код с использованием Just-In-Time (JIT) компилятора.

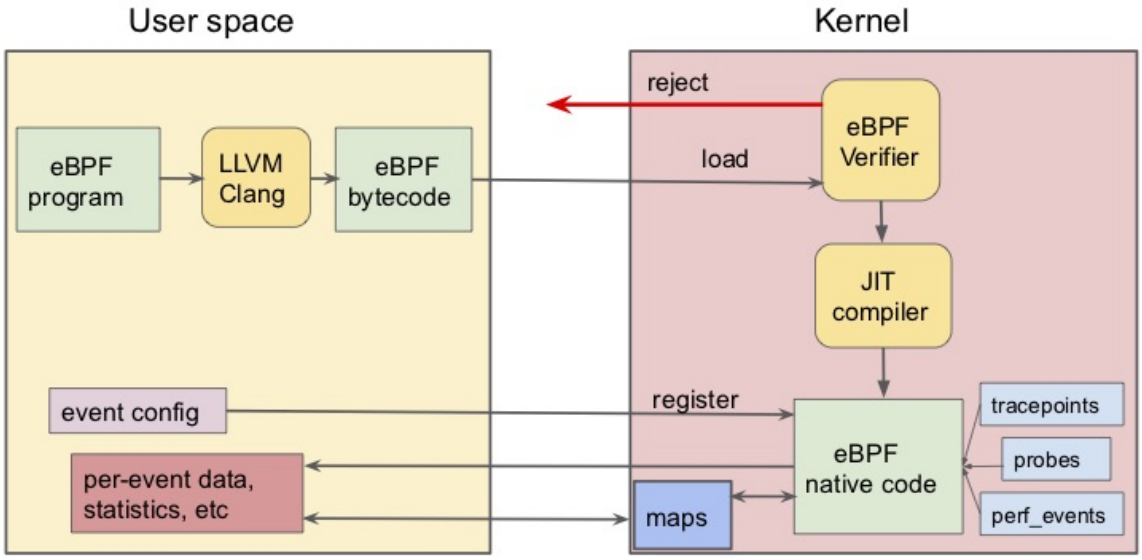

Рисунок 8 – Алгоритм технологии BPF/eBPF

и

Отследить путь сетевого пакета с помощью eBPF использованием два

типа:

— программы для работы в сети (Linux Traffic Control);

— программы kprobe (зондирования ядра);

— программы tracepoint (точки трассировки).

Программы для работы в сети позволяют контролировать сетевой трафик в своей системы, фильтровать и отбрасывать пакеты, поступающие от сетевого интерфейса. Разные программы могут работать по-разному связываться с различными этапами сетевой обработки пакетов в ядре или привязаны к сетевым событиям перед и после передачи или получения пакета. Например, программы сокетной фильтрации, привязывают BPF-программы к открытому сокету для получения доступа ко всем пакетам, чтобы наблюдать за информацией сетевого стека. Или программы eXpress Data Path (XDP), предоставляющие ограниченный набор информации из пакета, что у ядра было немного времени для ее обработки. Поскольку пакет исследуется и выполняется на ранней стадии, то контроль обработки его намного лучше. Реализуют несколько действий по управлению над пакетами и передают подсистеме ядра или игнорируют пакеты.

Программы kprobe, как говорилось ранее, функции, которые можно динамически подключать к определенным точкам вызова в ядре. Программы типа BPF kprobe позволяют использовать программы BPF в качестве обработчиков kprobe. Виртуальная машина BPF гарантирует, что программы kprobe всегда безопасны при запуске, что является преимуществом по сравнению с традиционными модулями kprobe.

Программы tracepoint, обеспечивающие информацию о поведение системы и оборудования, получаю доступ к области памяти и выполняют трассировку запущенных процессов. Программы tracepoint позволяют подсоединить BPF-программы к обработчику трассировки, предоставляемым ядром. Они менее гибки, чем kprobes, потому что должны быть определены ядром заранее, но гарантированно стабильны после введения в ядро соответствую-

щей точки отладки или модуля. Возможность прикреплять программы eBPF к точкам трассировки в дополнение к точкам тестирования ядра и пользовательских приложений позволяет отслеживать поведение приложений и системы во время выполнения.

Использование eBPF устраняет необходимость модификации исходного кода ядра и повышает возможности программного обеспечения по обеспечению безопасности при исполнении, то есть любая BPF-программа завершится без сбоев и программы не будут пытаться получить доступ к памяти вне области их деятельности, и реализации запуска на работающей систему. Но эти преимущества сопровождаются определенными ограничениями: программы имеют максимально допустимый размер, и циклы должны быть ограничены, чтобы гарантировать, что память системы никогда не будет исчерпана неправильно написанной программой BPF. Кроме того, при использовании eBPF наблюдается небольшое снижение производительности, поскольку программы пишутся в байт-коде BPF, а затем интерпретируются в ядре. Однако этот разрыв стал еще меньше теперь, когда eBPF поддерживает компиляцию JIT, что позволяет избавиться от фильтрации пакетов в каждой функции тем самым достичь низких накладных ресурсов. Данная технология позволяет разными программами выполнить требования к обработке сетевых пакетов. Изменение программ для фильтрации пакетов осуществляет работу при нужной нагрузке. eBPF не зависит от реализации ядра, что способствует независимости от конкретной сборки. Однако eBPF – молодой и пока еще развивающийся инструмент. В связи с этим в ходе разработки eBPF-программ могут возникать проблемы, связанные с отсутствием нужного инструментария, документации или её недостаточной информативностью.

2.2 Критерии сравнения методов сетевого мониторинга

В данном разделе будут описаны критерии, которые будут использоваться для сравнения подсистем и средств сетевого мониторинга ядра.

Таблица 1 – Критерии сравнения методов сетевого мониторинга

Критерий	**Описание**
Производительность	Работа при реальной нагрузки и низкие расход системных ресурсов.
Безопасность	Наличие гарантии, что внесенный код не вызовет сбой системы или нет необходимости к внедрению написанного кода
Скорость разработки	Быстрота разработки программ для сетевого мониторинга
Работоспособность	Запуск на работающей системе без сбоев или требования перезапуска
Гибкость	Возможность выполнить любые поставленные задачи
Независимость	Независимость от сборки ядра
Простота развертывания	Насколько сложно развертывать средства мониторинга на машине и сопровождением документации

2.3 Сравнение методов сетевого мониторинга

Таблица 2 – Сравнение методов сетевого мониторинга (Часть 1)

Критерий	**Утилиты**	**ftrace**	**BPF / eBPF**
Производительность	✓[1]	✓	✓
Безопасность	✓	✓	✓
Скорость разработки	—[2]	—	✓
Работоспособность	✓	✓	✓
Гибкость	✗[3]	✗	✗
Независимость	✓/✗[4]	✓	✓
Простота развертывания	✓	✗	✓

Таблица 3 – Сравнение методов сетевого мониторинга (Часть 1)

Критерий	**Модификация**	**kprobes**	**tracepoint**
Производительность	✓/✗	✓/✗	✓
Безопасность	✗	✓/✗	✓/✗
Скорость разработки	✗	✗	✓
Работоспособность	✓/✗	✓/✗	✓/✗
Гибкость	✗	✓	✓
Независимость	✗	✓/✗	✓
Простота развертывания	✓	✓/✗	✓/✗

[1]Метод сетевого мониторинга ядра Linux в реализации данного критерия полностью возможно.

[2]Метод сетевого мониторинга не требует изменение или добавление пользовательского кода в ядро Linux.

[3]Метод сетевого мониторинга ядра Linux в реализации данного критерия полностью невозможна либо значительно осложнена в сравнении с остальными методами.

[4]Метод сетевого мониторинга ядра Linux в реализации данного критерия возможно, но осложнена.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены:

— структура и принципы работы сетевой подсистемы ядра Linux;

— методы сетевого мониторинга ядра Linux или средства и подсистемы мониторинга сетевой подсистемы ядра Linux;

— критерии сравнения методов сетевого мониторинга;

— принципы работы методов касательно сетевой подсистемы;

— преимущества и недостатки каждого из методов.

Был выполнен обзор ядра Linux, его составных частей и сетевой подсистемы. Также проведен обзор и анализ существующих методов решений по сетевому мониторингу. Были сформирована критерии классификации методов сетевого мониторинга ядра Linux. Была проведена классификация методов сетевого мониторинга ядра Linux по критериям, сформированным в ходе работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Определение API [Электронный ресурс]. — Режим доступа: https://www.oxfordlearnersdictionaries.com/definition/english/api?q=API.

2 Количество багов версий ядра [Электронный ресурс]. — Режим доступа: https://bugzilla.kernel.org/report.cgi?bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&cumulate=0&y_axis_field=cf_kernel_version&width=1024&height=600&action=wrap&format=table.

3 Лав Роберт. Ядро Linux: описание процесса разработки, 3-е изд. — Москва: ООО «И.Д. Вильямс», 2013. с. 496.

4 О.И. Цилюрик. Модули Linux ядра: учебное пособие. — Казань: Казанский университет, 2011. С. 89 – 98.

5 Benvenuti C. Understanding Linux Network Internals. — Sebastopol: O'Reilly Media, Inc., 2005. p. 1064.

6 Гостев И. М. Операционные системы : учебник и практикум для вузов, 2-е изд. — Москва: Издательство «Юрайт», 2022. с. 164.

7 Структура сокета [Электронный ресурс]. — Режим доступа: https://www.kernel.org/doc/htmldocs/networking/API-struct-socket.html.

8 Хант Крэйг. TCP/IP Сетевое администрирование 3-е издание. — Санкт-Петербург–Москва: Издательство «Симво», Серия «Бест-селлеры O'Relly», 2008. С. 18 – 42.

9 Лора А. Чеппел Эд Титтел. TCP/IP: учебное пособие. — Санкт-Петербург: Издательство «БХВ-Петербург», 2003. С. 11 – 77.

10 Rosen Rami. Linux Kernel Networking Implementation and Theory. — Apress: ISBN., 2014. с. 648.

11 Мониторинг обработки IP-пакетов в ОС Linux [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/monitoring-obrabotki-ip-paketov-v-os-linux/viewer.

12 Probing the Guts of Kprobes [Электронный ресурс]. — Режим доступа: https://landley.net/kdocs/ols/2006/ols2006v2-pages-109-124.pdf.

13 Kernel Probes (Kprobes) [Электронный ресурс]. — Режим доступа: https://www.kernel.org/doc/html/latest/trace/kprobes.html.

14 Using the Linux Kernel Tracepoints [Электронный ресурс]. — Режим доступа: https://www.kernel.org/doc/html/latest/trace/tracepoints.html.

15 Declarative Tracepoints: A Programmable and Application Independent Debugging System for Wireless Sensor Networks [Электронный ресурс]. — Режим доступа: https://www.cs.virginia.edu/~stankovic/psfiles/tracepoints.pdf.

16 ftrace - Function Tracer [Электронный ресурс]. — Режим доступа: https://www.kernel.org/doc/html/latest/trace/ftrace.html.

17 Калавера Дэвид Фонтана Лоренцо. BPF для мониторинга Linux. — Санкт-Петербург: Издательство «Питер», Серия «Бест-селлеры O'Relly», 2021. с. 208.

ПРИЛОЖЕНИЕ А

Презентация к научно-исследовательской работе

Презентация содержит 15 слайдов.