ISBN 978-5-901795-13-2 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ. Переславль-Залесский, 2008

А. Ю. Михайлов

# Эффективный метод учета трафика в ОС Linux[*)]

Научный руководитель: к.т.н. Ю. В. Шевчук

Аннотация. В данной работе анализируются существующие методы учета сетевого трафика в ОС Linux. Предлагается новое эффективное решение, основанное на использовании возможностей подсистемы NetFilter ядра Linux версии 2.6.

## 1. Введение

Учет трафика, в общем случае, подразумевает процесс накопления информации о сетевом трафике, с последующим анализом и обобщением некоторых метаданных. Тип метаданных в основном зависит от конкретного приложения, но обычно включает в себя информацию о количестве пакетов, количестве байтов, IP-адресах отправителя и получателя. Существует множество причин для учета трафика, например:

- биллинг;
- мониторинг использования сети, информация о распределении нагрузки между частями сети;
- исследование распределения трафика между протоколами, поиск приложений, которые лучше утилизируют сеть.

Большинство компьютерных сетей нуждаются в некоторой форме учета трафика. Широко применяемая в современной сетевой инфраструктуре операционная система Linux имеет ряд механизмов, предназначенных для этой цели. Однако эти механизмы неадекватны современным объемам трафика либо не дают желаемой степени детализации.

---

[*)] Представлено по тематике: *Компьютерные сети и телекоммуникации, Программное обеспечение для компьютерных и сенсорных сетей.*

Накопление информации может быть реализовано несколькими путями. Традиционный метод — это захват всех пакетов в пространстве пользователя, где захват может быть выполнен несколькими механизмами: сокет `PF_PACKET`, `ipt_ULOG`, `ip_QUEUE` и многими другими. Затем эта информация анализируется в пространстве пользователя и агрегируется в потоки.

Поток, в общем случае, это набор пакетов, имеющих общие атрибуты. Фирма Cisco [1] определяет этот набор атрибутов 7 ключевыми полями [2]:

(1) IP-адрес источника;
(2) IP-адрес получателя;
(3) порт источника;
(4) порт получателя;
(5) номер протокола;
(6) бит ToS;
(7) входящий интерфейс.

Какие бы методы не использовались в данной схеме, здесь имеется мощное ограничение производительности: все пакеты должны быть переданы из пространства ядра в пространство пользователя и уже там проанализированы.

В данной работе предлагается принципиально новое решение, которое основано на использовании таблицы состояния соединений (connection tracking), находящейся внутри ядра Linux. Эта таблица должна присутствовать на любом маршрутизаторе и брандмауэре, поэтому накладные расходы минимальны.

Проблема анализа и обобщения информации сводится к передаче информации от маршрутизатора к другому компьютеру и обработке. В решении используется протокол NetFlow — стандарт де факто, разработанный Cisco, для сбора информации о IP–трафике. Маршрутизаторы (сенсоры) генерируют записи NetFlow, которые отправляются на обработку другим компьютерам (коллекторам). При таком подходе нам достаточно реализовать возможность генерации NetFlow пакетов на маршрутизаторах — для приема и обработки уже реализовано достаточно различных продуктов.

## 2. Существующие решения

Рассмотрим кратко некоторые популярные методы учета трафика.

### 2.1. nacctd

nacctd [3] является одним из самых старых инструментов для учета трафика. Принцип работы основан на использовании сокета AF_PACKET через библиотеку libpcap для захвата копий всех пакетов, приходящих на указанные интерфейсы. Затем производится разбор TCP/IP заголовков каждого пакета.

Поскольку решение основано на pcap, производительность nacctd падает из-за копирования каждого пакета из пространства ядра в пространство пользователя. Как решение, основанное на учете пакетов, а не потоков, производительность еще более проигрывает из-за того, что приходится разбирать каждый пакет.

### 2.2. ipt_LOG

Подсистема NetFilter содержит механизм для протоколирования нарушений заданных правил через циклический буфер сообщений ядра. Этот механизм называется ipt_LOG (или просто цель LOG). Эти сообщения затем обрабатываются klogd и syslogd, которые складывают их в один или несколько лог-файлов.

Так как ipt_LOG был смоделирован для передачи сообщений о нарушениях правил (не очень частые события), накладные расходы на использование этого механизма высоки. Каждый пакет должен быть интерпретирован в ядре, затем выведен в ASCII формате в циклический буфер сообщений, и уже потом скопирован от klogd к syslogd и еще раз скопирован в текстовый файл. Что еще хуже, syslogd, как правило, настроен на использование синхронной записи на диск.

Для обобщения и анализа обычно используются скрипты на Perl. Скрипт разбирает строки, полученные от ipt_LOG, строит список потоков, добавляет поле с информацией о размере пакетов и затем экспортирует в указанный формат.

### 2.3. ipt_ULOG

Цель ULOG является более эффективной версией цели LOG, описанной выше. Вместо того чтобы копировать ASCII сообщения через циклический буфер ядра, ipt_ULOG можно сконфигурировать для копирования только заголовков каждого пакета, а также для посылки этих копий сразу в большом количестве. Специальный процесс пространства пользователя ulogd получает эти частичные копии пакетов и занимается их дальнейшим разбором.

По сравнению с `ipt_LOG`, `ipt_ULOG` копирует меньше информации, а также требует меньшего количество переходов между пространствами ядра и пользователя, что ведет к улучшенной производительности. Тем не менее, этот механизм продолжает являться узконаправленным (передача пакетов, нарушивших некоторые правила) и поэтому имеет значительные накладные расходы на передачу заголовков всех пакетов в пространство пользователя.

### 2.4. ipac-ng

Каждое правило фильтрации пакетов в подсистеме NetFilter имеет два счетчика: количество пакетов и байтов, отвечающих данному правилу. Если составить список правил без цели, можно получить систему учета трафика (одно правило — один пользователь).

Существует несколько инструментов для разбора вывода `iptables` и снятия счетчиков. Наиболее часто используемая утилита — это `ipac-ng` [4], которая имеет много возможностей, например, помещение информации в базу данных SQL.

Этот подход достаточно эффективно работает для небольшого числа правил. Однако, иметь по одному счетчику для каждого порта каждого IP-адреса не получится из-за наличия огромного количества правил, применение которых будет занимать значительное время даже для одного пакета.

### 2.5. ipt_ACCOUNT

`ipt_ACCOUNT` [5] — специальная цель для `iptables`, не включенная в основное ядро, и поэтому требующая наложения заплатки. Эта цель хранит счетчик байтов для каждого IP-адреса в данной подсети, вплоть до ‘/8’. Эти счетчики могут быть прочитаны специальной утилитой `iptaccount`.

У данной разработки два серьезных ограничения: ограничение на размер подсети, и крупная гранула для подсчета — IP-адрес. Тем не менее, данное решение хорошо оптимизировано и достаточно эффективно.

### 2.6. ntop

`ntop` [6] — это датчик сетевого трафика, показывающий текущее использование сети. Это решение использует `libpcap` для захвата пакетов и агрегирует пакеты в потоки в пространстве пользователя. С общей точки зрения, `ntop` работает также как и `nacctd`.

Для улучшения производительности автор `ntop` реализовал механизм `PF_RING` — новую технику для захвата пакетов с нулевым копированием (zero-copy), поддержка которого была добавлена в `libpcap`. `PF_RING` — это очень большой шаг для улучшения производительности любого приложения, использующего `libpcap`.

В `ntop` также присутствует продукт `nProbe`, который экспортирует информацию о потоках в формате NetFlow.

Тем не менее, подход `ntop` также основан на пакетах. Каждый пакет должен быть разобран в пространстве пользователя, пусть даже никакого копирования и не используется.

## 3. Постановка задачи

Новое решение не должно обладать недостатками, которые присутствуют в описанных выше продуктах. То есть решение должно обладать следующими свойствами:

(1) должно быть основано на потоках: анализ каждого пакета — слишком дорого;
(2) захват пакетов и агрегация в потоки должны производиться в пространстве ядра: частые переходы между пространствами ядра и пользователя — слишком дорого;
(3) результат вывода должен быть стандартизован: свой формат не нужен.

Основная идея для нового метода — это использование таблицы состояния соединений, являющейся частью подсистемы NetFilter, что дает нам следующие преимущества:

(1) захват пакетов, обработка и агрегация в потоки уже есть;
(2) NetFilter является частью любого маршрутизатора и брандмауэра, основанного на Linux, поэтому накладные расходы за поддержание этой таблицы уже оплачены.

Тем не менее, требуется решение еще нескольких проблем. Во-первых, это экспорт данных из таблицы состояний, во-вторых, заполнение пакета NetFlow, для чего требуется расширение таблицы и, в-третьих, проблема производительности — решение не должно иметь явных ограничений.

## 4. Разработка нового метода

Приступим к описанию нового решения. Сразу стоит отметить, что наше решение функционирует полностью в пространстве ядра, никакого общения с пространством пользователя не совершается — однако, некоторое взаимодействие возможно и об этом будет упомянуто позднее.

### 4.1. Экспорт данных из таблицы состояний

Экспорт потока должен происходить при его удалении из таблицы. Для реализации этого используется технология оповещения об изменении таблицы состояний (conntrack notifier). Данная технология использует базовую инфраструктуру оповещения ядра (struct notifier_ block) для передачи информации об изменениях в таблице другим частям ядра. Эта технология позволяет нам обойти классическую схему поллинга (polling). Если нам интересны только события, связанные с удалением, то достаточно зарегистрировать функцию, которая будет вызываться для каждой удаленной записи.

Однако, существует другая причина для экспорта, которая особенно актуальна для биллинга. Продолжительный поток может содержать большой объем переданной информации; если данные потока будет учитываться только по его завершению, то мы не можем гарантировать актуальность текущего состояния счета абонента. То есть должна существовать возможность для временного экспорта информации.

С этой целью была реализована новая цель для iptables `NF_TIMEOUT`. Например, при помощи команды

```
iptables -A FORWARD -p tcp -s 192.168.0.0/24 -j NF_TIMEOUT \
--nfsecs 1200 --nfbytes 10000000
```

можно указать, что экспортировать поток для клиентов из подсети 192.168.0.0/24 нужно через 20 минут, если объем информации в потоке превысит 10 мегабайт.

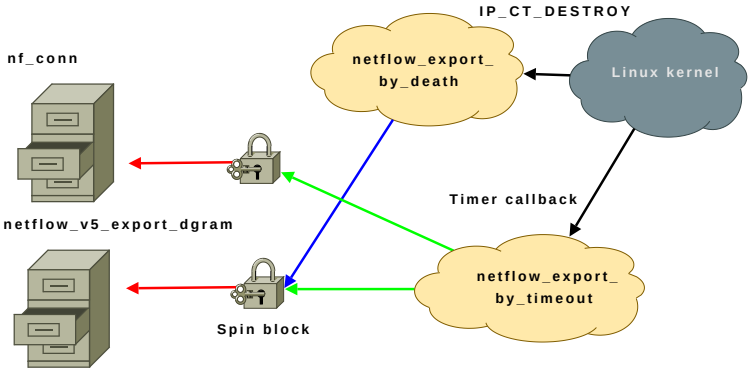


Рис. 1. Структура блокировок модуля

Для минимазации времени экспорта запись не удаляется целиком — модуль просто перехватывает блокировку основной таблицы соединений `nf_conn` и очищает счетчики. Общая структура блокировок показана на рис. 1. `netflow_v5_export_dgram` — статически выделяемая структура записи (заголовок+содержимое) NetFlow. Статическое выделение памяти и спиновая блокировка используются для повышения производительности решения.

### 4.2. Записи NetFlow и записи таблицы состояний соединений

На рис. 2 показана структура записи NetFlow. Зеленым цветом отмечены поля, которые заполнить легко — информация уже имеется в таблице состояний соединений. Желтым цветом отмечены поля, информации о которых мы не имеем, но возможно их заполнить путем не очень сложных модификаций ядра. Красным цветом отмечены поля, которые заполнять или сложно, или нецелесообразно. Рассмотрим все эти поля подробнее.

IP адреса и порты отправителя и источника, количество пакетов и байтов, используемый IP протокол уже имеются в таблице — их достаточно просто скопировать.

Время начала и конца потока (Start/End time of flow) получаются достаточно просто: нужно добавить время начала потока в таблицу,

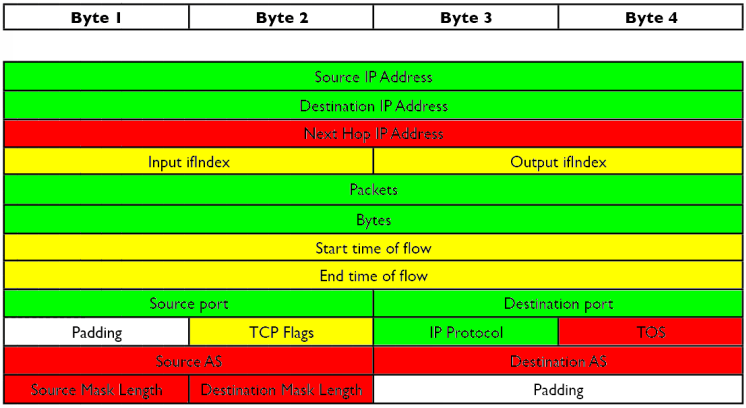


Рис. 2. Структура записи NetFlow

а время конца заполнить при экспорте, в случае временного экспорта — обновляем время начала потока текущим.

Поле TCP flags (флаги TCP) получается выполнением операции ИЛИ на все биты TCP соединения, и также несложно восстанавливается по информации, имеющейся в таблице.

Несколько сложнее все обстоит с номерами входящих/исходящих интерфейсов, так как это относится к более низкому уровню, чем тот, на котором оперирует таблица состояний. Выбор маршрутов и интерфейсов — это отдельный сетевой код ядра, выполняющий маршрутизацию, в то время как в таблице отслеживаются состояния текущих соединений. Мы можем предположить, что если пакет был послан на некоторый интерфейс, то и ответ придет на этот же интерфейс. Это верно в большом количестве случаев, но не всегда. Пример: мы посылаем SNMP запрос на интерфейс eth0, получаем ответ с eth1, в то время как еще один ответ приходит с eth2 — это вполне правдоподобный сценарий. В решении реализована относительно простая схема: разбираем буфер пакета, инициирующего поток, и запоминаем индекс интерфейса. Если поток переходит в состоянии отвеченного, то берем первый пришедший пакет и таким же образом извлекаем индекс.

Имея индекс интерфейса и целевой адрес мы также можем заполнить еще одно поле — Next Hop IP Address (IP адрес компьютера на следующем хопе). Для его заполнения достаточно узнать IP-адрес

шлюза для данного IP-адреса, для чего модуль взаимодействует с сетевым кодом ядра, отвечающим за маршрутизацию.

Значение поля TOS (Type of Service) не относится к потоку в том смысле, что это свойство пакета, но не потока. Оборудование Cisco заполняет это значение, используя первый пакет потока, поэтому в описываемом решении мы используем такую же реализацию.

Поля Source/Destination Mask Prefix (длина префикса входящего/исходящего IP-адреса) и Source/Destination AS Number (номер автономной системы, отвечающей входящему/исходящему IP-адресу) используются для BGP-маршрутизации и специфических видов агрегации, поэтому эти поля заполнять не целесообразно.

### 4.3. Экспорт записи NetFlow

Для экспорта NetFlow, по стандарту, используются протоколы UDP и SCTP. На данный момент для экспорта используется UDP и пакет отсылается из пространства ядра. С другой стороны, при использовании UDP доставка не гарантирована. Однако, потерю пакета обнаружить можно. В заголовке пакета NetFlow присутствует поле Flows Seen — это номер последнего экспортированного потока. Если значение этого поля в одной из пришедших записей $N$, а в следующей $N + k$, где число $k$ меньше количества потоков, содержащихся во второй записи, то можно констатировать факт потери. В планах на будущее остается реализация механизма гарантированной доставки — для этого возможна кооперация с пространством пользователя, так как реализовывать этот механизм в ядре не рационально. Также в планах на будущее остается защита экспортируемой информации от перехвата злоумышленником.

### 4.4. Механизм ожидаемых соединений и его применение для учета трафика

Часто можно увидеть сильно связанные соединения, потоки которых различаются. Пример: протокол FTP в пассивном режиме использует порт 21 для посылки команд серверу, но для получения файла используется динамически назначаемый порт в диапазоне от 1024 до 65536. То есть два независимых соединения тесно связаны на уровне приложения.

В таблице состояний реализован механизм, основанный на “помощниках” (helpers) — программном коде, который дает системе возможность понять, связаны ли два соединения. Для этого вводится

понятие ожидаемого соединения — это соединение, появления которого система ожидает определенное время. То есть по данному потоку строится шаблон, и в случае если новое соединение будет ему удовлетворять, их можно считать связанными.

Все это дает возможность находить объединять некоторое количество потоков семантически, для учета трафика это означает более простой способ анализа. Места для подобной информации в записи NetFlow, к сожалению, нет, но можно использовать зарезервированные поля пакета NetFlow (Padding).

### 4.5. Обнаружение подозрительной сетевой активности

Рассмотрим одно из распространенных применений учета трафика — обнаружение подозрительной сетевой активности (атаки типа DDoS, вирусы). Общая идея предлагаемого метода для обнаружения этих аномалий также базируется на использовании подсистемы NetFilter. Используя технику оповещения об изменениях в таблице, упомянутую ранее, мы можем выполнять действие для каждого изменения таблицы. Для каждого изменения мы получаем информацию о потоке: IP-адреса, объем (количество пакетов и байтов), статус (отвеченный или нет). Используя эту информацию, можно накапливать данные о сетевой активности и делать некоторые выводы. При таком подходе, достаточно описать само “действие” для каждого из вида атак.

Рассмотрим конкретный пример. DDoS-атака — это разновидность атаки злоумышленника на компьютерную систему, целью которой является создание таких условий, при которых легитимные пользователи системы не могут получить доступ к предоставляемым ресурсам. Очень часто этот вид атаки характеризуется наличием быстрых потоков однотипных пакетов на некоторый адрес. Также эти потоки являются, как правило, однонаправленными.

В простейшем случае, “действие” можно описать как проверку — является поток отвеченным или нет, если нет — то проверить его объем. Если он больше некоторого числа, тогда можно сделать предположение о подозрительной активности. Выбор данного числа представляет одну из проблем: в самом простом случае можно указать наперед заданное число, но тогда система будет часто делать ложные выводы. Один из выходов — использование самоадаптирующихся алгоритмов: то есть выбирать это число, исходя из текущей сетевой активности.

Можно предложить и другие улучшения, например, имеет смысл агрегировать потоки по IP-адресу источника, то есть накапливать информацию о каждом из пользователей сети. Если IP-адрес часто порождает подозрительные потоки, то логично делать вывод о подозрительном поведении не потока, а уже пользовательского окончания.

## 5. Результаты

В работе были кратко проанализированы наиболее распространенные методы учета трафика в Linux. Было предложено и реализовано новое эффективное решение.

Описанный модуль распространяется в виде заплатки к ядру и утилите конфигурирования фильтра пакетов iptables и добавляет две новые опции для компиляции: базовую функциональность для выполнения роли NetFlow сенсора и возможность промежуточного экспорта потоков.

## Список литературы

[1] Cisco Systems, Inc. — `http://www.cisco.com`.

[2] Netflow Solution Guide. — `http://www.cisco.com/en/US/docs/ios/solutions_docs/netflow/nfwhite.html`.

[3] Exorsus.net: Net-Acct. — `http://exorsus.net/projects/net-acct/`.

[4] Sourceforge.net: ipac-ng. — `http://sourceforge.net/projects/ipac-ng/`.

[5] Intra2net: IPT_account, `http://www.intra2net.com/de/produkte/opensource/ipt_account/`.

[6] Ntop: network top, `http://www.ntop.org/`.

A. Y. Mikhailov. *Efficient method of network accounting in OS Linux* // Proceedings of Programm Systems institute scientific-practical conference “Program systems: Theory and applications”, devoted to the $15^{\text{th}}$ anniversary of Pereslavl University named A. K. Ailamazyan. — Pereslavl-Zalesskij, 2008. — p. 163 — 173. — ISBN 978-5-901795-13-2 *(in Russian).*

Abstract. In this paper existing methods of network accounting are analyzed. New efficient solution based on NetFilter subsystem of Linux 2.6 kernel is proposed.

*Перевод проверен:* к.т.н. Ю. В. Шевчук