электронное научно-техническое издание

НАУКА и ОБРАЗОВАНИЕ

Эл № ФС77 - 30569. Государственная регистрация №0421100025. ISSN 1994-0408

Мониторинг обработки IP-пакетов в ОС Linux

77-30569/342083

04, апрель 2012

А.О. Крючков, В.А. Крищенко

УДК 004.451:004.7

МГТУ им. Н.Э. Баумана
alexey.kryuchkov.mx@gmail.com, kva@bmstu.ru

Введение

При изучении работы компьютерной сети, использующей протокол IP [1, 2], и выявлении неисправностей интересна информация об обработке отдельных IP-пакетов на узле сети. Инструменты, подобные Tcpdump [3], позволяют зафиксировать прохождение пакета через сетевой интерфейс, но не дают информации о событиях, случившихся с пакетом в ядре ОС — фрагментации, обработке сетевым экраном и тому подобных. Тем не менее, потребность в получении такой информации существует.

Открытость ОС Linux позволяет реализовать инструмент получения информации о событиях, происходящих в сетевой подсистеме ядра ОС, путем модификации исходного кода ядра. В настоящей статье описывается создание такого инструмента.

Статья организована следующим образом. В разд. 1 в общем виде описывается структура системы мониторинга обработки IP-пакетов. В разд. 2 приводится обзор функций сетевой подсистемы ОС Linux, выполняющих обработку IP-пакетов. В разд. 3 обсуждается размещение в коде этих функций ловушек, фиксирующих события, а в разд. 4 приводится алгоритм предварительной обработки зафиксированных событий. Наконец, в разд. 5 приводится пример результата работы системы мониторинга обработки IP-пакетов.

1. Структура системы слежения за обработкой пакетов

Структурная схема разрабатываемой системы представлена на рис. 1. Поскольку обработкой сетевых пакетов занимается сетевая подсистема ОС Linux, естественно разместить код, перехватывающий события («ловушки»), в самой сетевой подсистеме. Конечным потребителем информации о событиях является некоторое пользовательское приложение, не рассматриваемое детально в данной работе; в простейшем варианте оно может просто выводить получаемую информацию на экран. Модуль регистрации событий, оформляемый как динамически загружаемый модуль ядра ОС, занимается предварительной обработкой перехваченных событий и подготовкой их к передаче пользовательскому приложению.

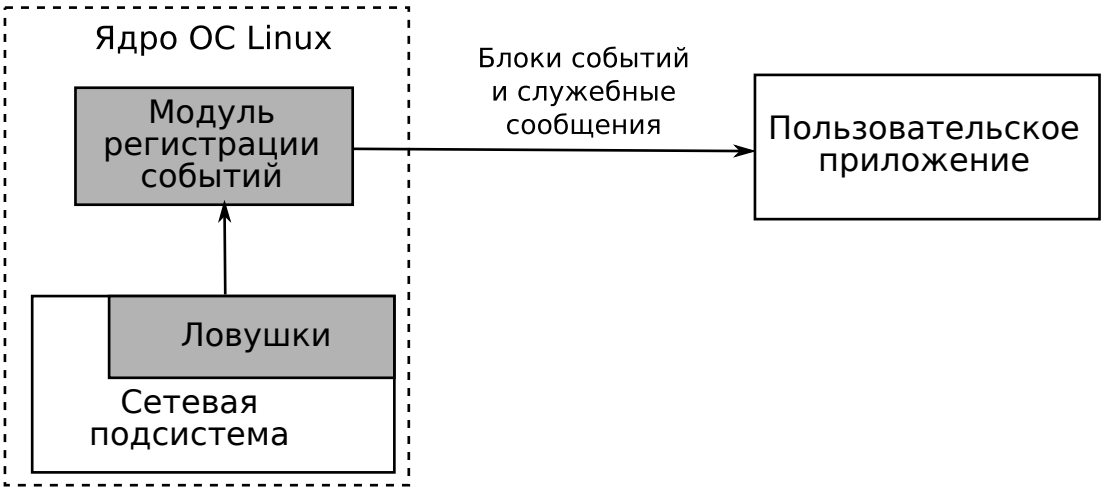

Рис. 1. Структурная схема системы слежения

2. Обработка IP-пакетов в сетевой подсистеме ОС Linux

Для перехвата информации о событиях обработки IP-пакета в сетевой подсистеме ядра ОС необходимо расположить ловушки. На рис. 2 представлена схема взаимодействия функций ядра, образующих основу реализации протокола IPv4; реализация протокола IPv6 в целом аналогична [4]. «Функции» с префиксом **NF_** на схеме фактически реализуются функцией **nf_hook_slow** сетевого экрана Netfilter, являющимся компонентом ядра Linux.

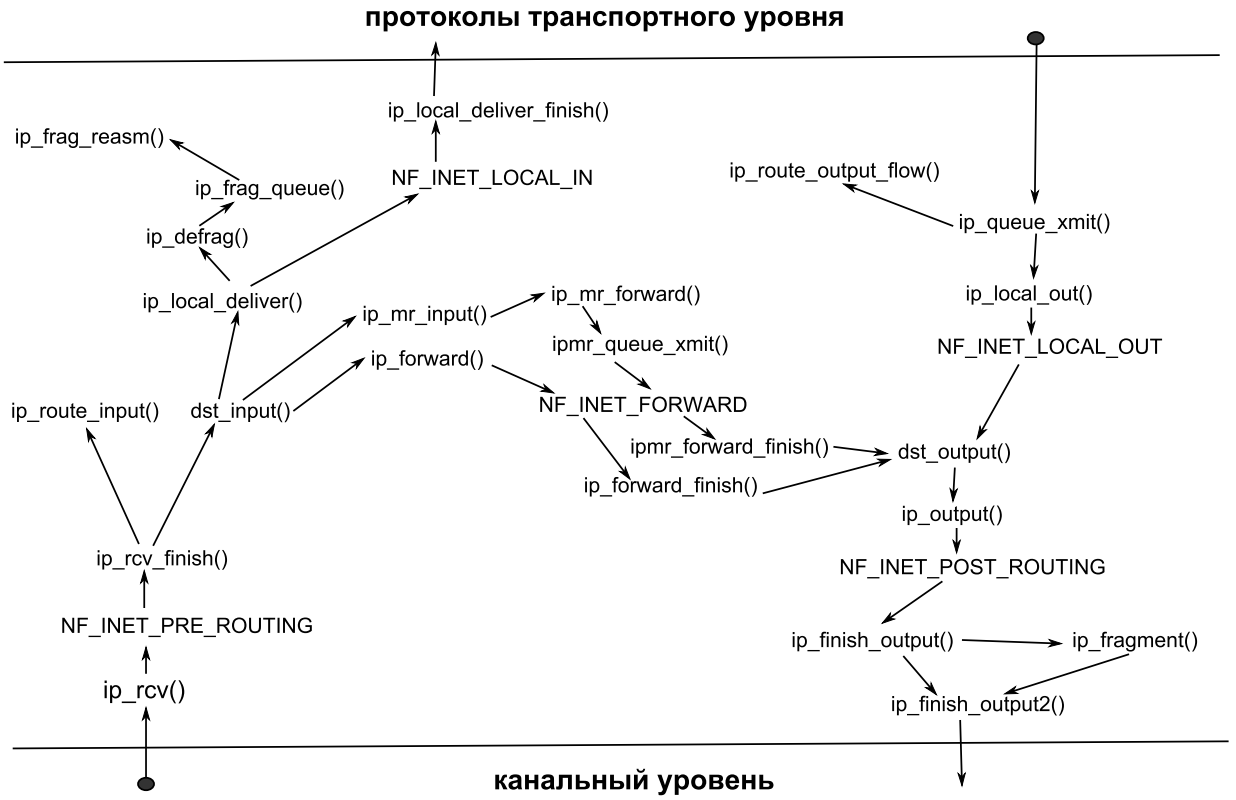

Рис. 2. Функции сетевой подсистемы ядра ОС Linux

В табл. 1 приведено соответствие между событиями обработки IP-пакета и функциями сетевой подсистемы ядра Linux.

Таблица 1

Связь событий обработки пакетов и функций ядра Linux

Событие	Реализация IPv4	Реализация IPv6
Формирование нового IP-пакета узлом-отправителем	ip_local_out	ip6_local_out
Доставка пакета по назначению	ip_local_deliver_finish	ip6_input_finish
Передача пакета сетевому интерфейсу	ip_finish_output2	ip6_output_finish
Поступление пакета от сетевого интерфейса	ip_rcv	ipv6_rcv
Пересылка пакета (простая и многоадресная)	ip_forward, ip_mr_forward	ip6_forward, ip6_mr_forward
Отбрасывание пакета	любая функция + kfree_skb	
Фрагментация пакета	ip_fragment	ip6_fragment
Дефрагментация пакета	ip_defrag, ip_frag_reasm	ip6_frag_rcv, ip6_frag_reasm
Преобразование сетевых адресов	nf_hook_slow в Netfilter	

3. Размещение ловушек в сетевой подсистеме

Ловушки, размещаемые нами в коде функций сетевой подсистемы, представляют собой вызовы функции-обработчика перехваченных событий. Далее по тексту мы опишем алгоритм работы этой функции.

Каждая ловушка сигнализирует о конкретном событии с конкретным IP-пакетом, поэтому в обработчик перехваченных событий передается идентификатор IP-пакета, место возникновения события, код события и зависящие от события дополнительные параметры. В качестве идентификатора IP-пакета используется адрес экземпляра структуры ядра **sk_buff**, в которой содержится IP-пакет. Событие идентифицируется функцией, где оно возникло, и кодом из множества –GENERIC, BEFORE, AFTER, DROPPED″, поскольку в ряде функций возникает несколько событий. Таким образом, функция-обработчик имеет следующий прототип:

```
void event_handler(struct sk_buff *skb,
    int event_source, int code,
    void* extra_params);
```

Ловушка с кодом GENERIC размещается в самом начале каждой функции, показанной на рис 2. Ловушки с кодом DROPPED размещаются в местах отбрасывания пакетов, перед вызовом функции **kfree_skb**. Дополнительная ловушка с кодом DROPPED размещается в функции **kfree_skb** для перехвата всех случаев удаления пакета.

Поясним общий случай размещения ловушек в функции следующим псевдокодом:

```
function f(struct sk_buff *skb, ...) {
    event_handler(skb, F, GENERIC, 0);
    ...
    if (обнаружены ошибки)
        goto out;
    ...
    return;
out:
    event_handler(skb, F, DROPPED, 0);
    kfree_skb(skb);
}
```

Использование ловушек с кодами BEFORE и AFTER специфично для конкретных функций. В функции **nf_hook_slow** размещаются две ловушки: одна — до применения правил сетевого экрана, вторая — после. Их расположение показано в следующем псевдокоде:

```
function nf_hook_slow(struct sk_buff *skb, uint hook, ...) {
     event_handler(skb, hook, BEFORE, 0);
     вердикт = "пропустить";
     for each (rule_chain in netfilter_chains[hook]) {
         вердикт = rule_chain(skb, вердикт);
         if (вердикт != "пропустить")
            break;
     }
     if (вердикт == "пропустить") {
         event_handler(skb, hook, AFTER, 0)
     } else {
         event_handler(skb, hook, DROPPED, 0);
         kfree_skb(skb);
     }
}
```

Фрагментация пакета (функция **ip_fragment**) сопровождается тремя ловушками, как показано в следующем псевдокоде:

```
function ip_fragment(struct sk_buff *skb) {
    event_handler(skb, IP_FRAGMENT, BEFORE, 0);
    while (нужно разделять на фрагменты) {
        struct sk_buff *skb_frag = создать_очередной_фрагмент(skb);
        event_handler(skb_frag, IP_FRAGMENT, GENERIC, skb);
        ip_finish_output2(skb_frag);
    }
    event_handler(skb, IP_FRAGMENT, AFTER, 0);
}
```

Аналогично размещаются ловушки в функции многоадресной пересылки **ip_mr_forward**, где создаются копии исходного пакета. В функции дефрагментации **ip_frag_reasm** ловушки размещаются следующим образом:

```
function ip_frag_reasm(struct ipq *qp, struct sk_buff *skb) {
    выделить_память_под_собираемый_пакет(skb)
    for each (skb_frag in qp) {
        event_handler(skb_frag, IP_FRAG_REASM, BEFORE, skb);
        добавить_данные_к_собранной_части(skb_frag, skb);
        kfree_skb(skb_frag);
    }
    event_handler(skb, IP_FRAG_REASM, AFTER, 0);
}
```

4. Обработка перехваченных событий

Функция-обработчик событий **event_handler** содержится в модуле регистрации событий. Задача этого модуля — подготовить данные о событиях к передаче пользовательскому приложению. Модуль выполняет некоторую предварительную обработку перехваченных событий, суть которой описана ниже.

Во-первых, события, связанные с отдельными пакетами, группируются в *блоки событий*. Это сделано с целью понизить частоту обмена данными с пользовательским приложением. Новый блок событий создается при появлении в сетевой подсистеме ОС нового IP-пакета или фрагмента, и передается пользовательскому приложению в тот момент, когда пакет прекращает свое существование.

Во-вторых, модуль фиксирует осуществление преобразования сетевых адресов (NAT). Для этого сравнивается заголовок пакета в состояниях до и после обработки сетевым экраном Netfilter.

В-третьих, помимо блоков событий, модуль регистрации событий генерирует и передает пользовательскому приложению два типа служебных сообщений — *уведомления о поступлении пакета из сети* и *уведомления о завершении операции* (фрагментации или многоадресной рассылки). Пример последовательности отправки сообщений при обработке одного IP-пакета показан на рис. 3.

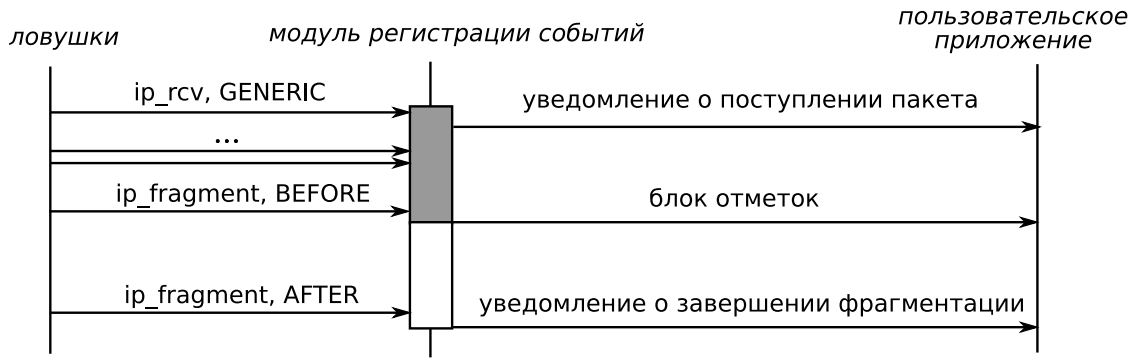

Рис. 3. Последовательность отправки сообщений модулем регистрации событий

Таблица 2

Правила обработки событий

Источник события	Код события	Действия
ip_rcv	GENERIC	Отправить уведомление о получении пакета; создать новый блок событий
ip_local_out ip_fragment ip_mr_forward ip_frag_reasm	GENERIC GENERIC GENERIC AFTER	Создать новый блок событий для обрабатываемого IP-пакета
ip_fragment ip_mr_forward	AFTER AFTER	Отправить уведомление о завершении операции
ip_local_deliver_finish ip_finish_output2 любой ip_fragment ip_mr_forward ip_frag_reasm	GENERIC GENERIC DROPPED BEFORE BEFORE BEFORE	Закрыть блок событий и отправить его пользовательскому приложению
NF_INET_PRE_ROUTING NF_INET_LOCAL_OUT NF_INET_POST_ROUTING	BEFORE BEFORE BEFORE	Зафиксировать значения адресов и портов для обнаружения преобразования сетевых адресов
NF_INET_PRE_ROUTING NF_INET_LOCAL_OUT NF_INET_POST_ROUTING	AFTER AFTER AFTER	При отличии текущих значений от зафиксированных добавить в блок событие преобразования адресов

Модуль регистрации событий ведёт список незаконченных блоков событий для пакетов, находящихся в данный момент в сетевой подсистеме узла. Вызываемая в ловушке функция ищет в данном списке блок по перехваченному адресу структуры **sk_buff** и добавляет в него информацию о событии. Дальнейшие действия модуля регистрации приведены в табл. 2.

5. Пример работы системы мониторинга

Ниже приведен листинг вывода пользовательского приложения, полученный при проведении простейшего опыта: через компьютер, выступающий в роли маршрутизатора, проходит один IP-пакет, разбитый на два фрагмента.

```
01  PACKET RECEIVED: D085580 from eth1
02  PACKET RECEIVED: D085680 from eth1
03  EVENT BLOCK for D085580
04  1  ip_rcv GENERIC
05  2  NF_INET_PRE_ROUTING BEFORE
06  3  ip_defrag GENERIC
07  4  ip_frag_reasm BEFORE (reasm. target: D085680)
08  END
09  EVENT BLOCK for D085680
10  1  ip_rcv GENERIC
11  2  NF_INET_PRE_ROUTING BEFORE
12  3  ip_defrag GENERIC
```

```
13  4  ip_frag_reasm BEFORE (reasm. target: D085680)
14  END
15  EVENT BLOCK for D085680
16  1  ip_frag_reasm AFTER
17  2  NF_INET_PRE_ROUTING AFTER
18  3  ip_forward GENERIC
19  4  NF_INET_FORWARD BEFORE
20  5  NF_INET_FORWARD AFTER
21  6  ip_finish_output GENERIC
22  7  ip_fragment BEFORE
23  END
24  EVENT BLOCK for D085680
25  1  ip_fragment AFTER (is a fragment of D085680)
26  2  ip_finish_output GENERIC (iface: eth0)
27  END
28  EVENT BLOCK for D085780
29  1  ip_fragment AFTER (is a fragment of D085680)
30  2  ip_finish_output GENERIC (iface: eth0)
31  END
32  OPERATION COMPLETED: fragmentation of D085680
```

Строки 01-02 показывают два уведомления о поступлении пакетов (фрагментов) из сети. В них указываются идентификаторы пакетов — адреса экземпляров структур **sk_buff**.

Строки 03-08 описывают блок событий для фрагмента D085580. Блок оканчивается событием дефрагментации. Строки 09-14 содержат аналогичный блок для второго фрагмента, имеющего адрес D085680.

В строках 15-23 описывается жизнь собранного из фрагментов пакета. Заметим, что сетевая подсистема использовала для собранного пакета буфер одного из фрагментов (D085680), поэтому их адреса совпадают. Тем не менее, разработанная система мониторинга четко различает собранный пакет и его фрагменты.

Блок событий для собранного пакета завершается событием фрагментации. При этом порождаются два фрагмента, размещенные по адресам D085680 (опять-таки используется буфер исходного пакета) и D085780. Строки 24-32 описывают блоки событий для этих фрагментов, заканчивающиеся передачей в сетевой интерфейс eth0.

Последняя строка листинга показывает уведомление о завершении фрагментации пакета D085680. Оно оначает, что цикл разбиения на фрагменты закончен и новых фрагментов этого же пакета не последует.

Заключение

В работе были описаны модификации ядра ОС Linux, позволяющие получать полную информацию о получении, обработке и пересылке IP-пакетов. Приведено описание инструмента мониторинга, основанного на описанных модификациях, и пример его использования.

Список литературы

1. *Postel J.* Internet Protocol. RFC 791 (Standard). 1981.
2. *Deering S.* Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard). 1998.
3. *Mccanne St.*, *Jacobson V.* The BSD Packet Filter: A New Architecture for User-level Packet Capture // Proc. 1993 Winter USENIX Conference, 1993. P. 259–269.
4. *Wehrle K.*, *Pahlke F.*, *Ritter H.* Linux Network Architecture. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004. 648 p.

electronic scientific and technical periodical

SCIENCE and EDUCATION

El № FS77 - 30569. №0421100025. ISSN 1994-0408

IP packet processing monitoring in Linux

77-30569/342083

 # 04, April 2012

A.O. Kryuchkov, V.A. Krishchenko

Bauman Moscow State Technical University
alexey.kryuchkov.mx@gmail.com, kva@bmstu.ru

The paper discusses a method to retrieve detailed information on IP packet processing in Linux operating system. Modifications to Linux networking subsystem source code are described, along with the architecture of a monitoring tool based on the method being discussed. An example of the monitoring tool's output is provided. Possible applications of the method include troubleshooting and studying of computer networks.

References

1. *Postel J.* Internet Protocol. RFC 791 (Standard). 1981.
2. *enDeering S.* Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard). 1998.
3. *Mccanne St.*, *Jacobson V.* The BSD Packet Filter: A New Architecture for User-level Packet Capture // Proc. 1993 Winter USENIX Conference, 1993. P. 259–269.
4. *Wehrle K.*, *Pahlke F.*, *Ritter H.* Linux Network Architecture. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004. 648 p.