электронное научно-техническое издание

НАУКА и ОБРАЗОВАНИЕ

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

Метод отслеживания сетевых пакетов

77-30569/400433

 # 04, апрель 2012

А.О. Крючков, В.А. Крищенко

УДК 004.728.4

МГТУ им. Н.Э. Баумана
kva@bmstu.ru

Введение

Задача отслеживания отдельных пакетов в компьютерных сетях, использующих протокол IP [1, 2], является достаточно актуальной и встречается при выявлении неполадок и изучении работы самого протокола. В идеале хотелось бы получать для каждого пакета его полный жизненный цикл, от создания пакета и до доставки получателю, с информацией об отдельных событиях — прохождении через маршрутизатор, фрагментации и дефрагментации, выполнении многоадресной маршрутизации, туннелировании, преобразовании сетевых адресов, отбрасывании пакета.

Существующие средства позволяют решать эту задачу лишь частично. Так, анализаторы трафика или снифферы, такие как Tcpdump [3], ограничиваются лишь фиксацией прохождения пакета через сетевой интерфейс, и не дают информации о событиях, случившихся с пакетом в ядре ОС. Утилита Traceroute, использующая служебный протокол ICMP, позволяет определять маршрут только специально созданных IP-пакетов, и её нельзя использовать для отслеживания пакетов обычного TCP-соединения.

В данной статье описывается создание метода отслеживания пакетов, позволяющего полностью решить поставленную задачу получения детальной информации о жизненном цикле пакетов. Предполагается, что необходимое программное обеспечение может быть установлено на каждом из узлов анализируемой сети, включая маршрутизаторы.

Статья организована следующим образом. В разд. 1 формализуется описание жизненного цикла пакета, которое требуется получить как результат работы разрабатываемого метода. В разд. 2 приводится общее описание этапов метода отслеживания, а в разд. 3 приводятся ключевые алгоритмы. Раздел 4 посвящен описанию структуры программного комплекса, реализующего разрабатываемый метод. Наконец, в разд. 5 приводится опыт с таким программным комплексом.

1. Формализация жизненного цикла IP-пакета

На рис. 1 приведена классификация событий, составляющих жизненный цикл отдельного IP-пакета.

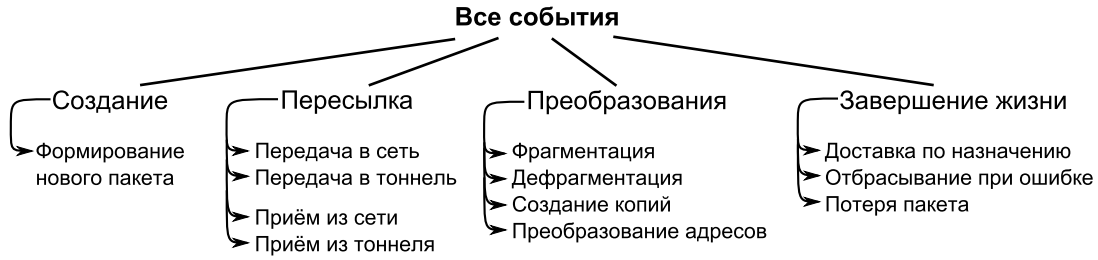

Рис. 1. Классификация событий жизненного цикла IP-пакетов

Структуру данных для хранения информации о жизненном цикле одного IP-пакета будем называть *журналом*. Журнал состоит из *блоков*, которые подразделяются на блоки событий и служебные блоки. *Блоки событий* описывают движение IP-пакета в рамках одного узла сети и содержат список событий. *Служебные блоки* могут быть трёх видов:

1) блоки, связывающие несколько пакетов отношениями «один-ко-многим» — блоки фрагментации, дефрагментации, многоадресной рассылки;

2) блоки входа и выхода из тоннеля, содержащие ссылки на журнал пакета, послуживший транспортом в тоннеле;

3) блоки-терминаторы, завершающие журнал и связанные с событиями завершения жизни пакета.

В журнале блоки связаны между собой дугами, отражающими логическую последовательность событий. Таким образом, журнал представляет собой ациклический направленный граф, где вершинами являются блоки, относящихся к одному исходному IP-пакету (рис. 2).

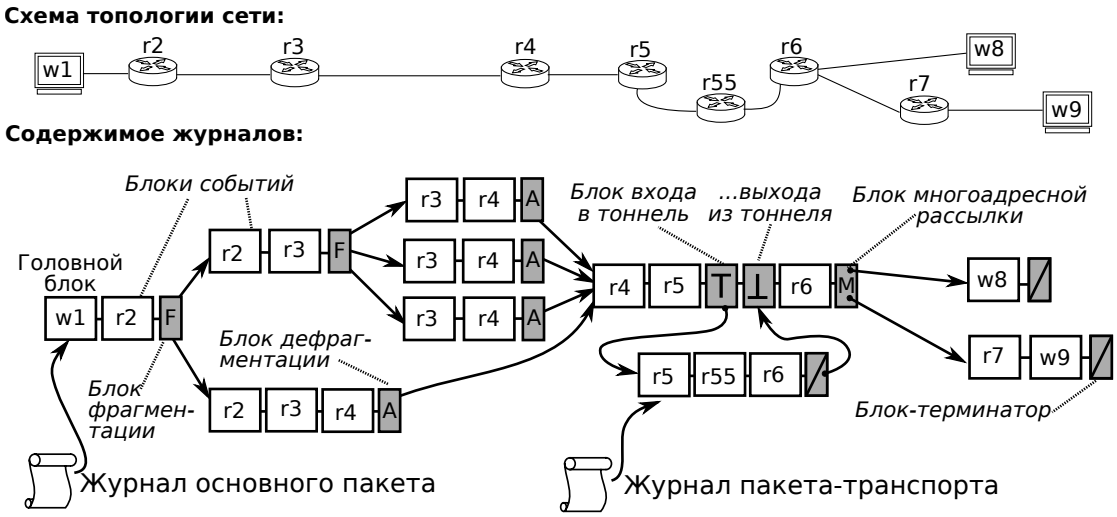

Рис. 2. Пример журнала

Блок событий, соответствующий только что сформированному IP-пакету, называется *головным блоком* и является единственным истоком — вершиной графа, в которую не входит ни одна дуга. Граф может содержать произвольное число стоков — блоков-терминаторов.

В примере на рис. 2 исходный пакет разбивается на два фрагмента на узле **r2**. Затем первый фрагмент разбивается еще на три на узле **r3**. Далее фрагменты собираются на узле **r4**, собранный пакет проходит через тоннель, после чего маршрутизатор **r6** выполняет многоадресную маршрутизацию, передавая две копии пакета узлам **w8** и **r7**.

2. Этапы метода отслеживания IP-пакетов

В разрабатываемом методе журналы движения пакетов формируются в несколько этапов (рис. 3):

1) перехват событий, возникающих в сетевой подсистеме ядра ОС;

2) предварительная обработка перехваченных событий и создание блоков событий для сборки журнала;

3) соединение блоков в готовые журналы IP-пакетов.

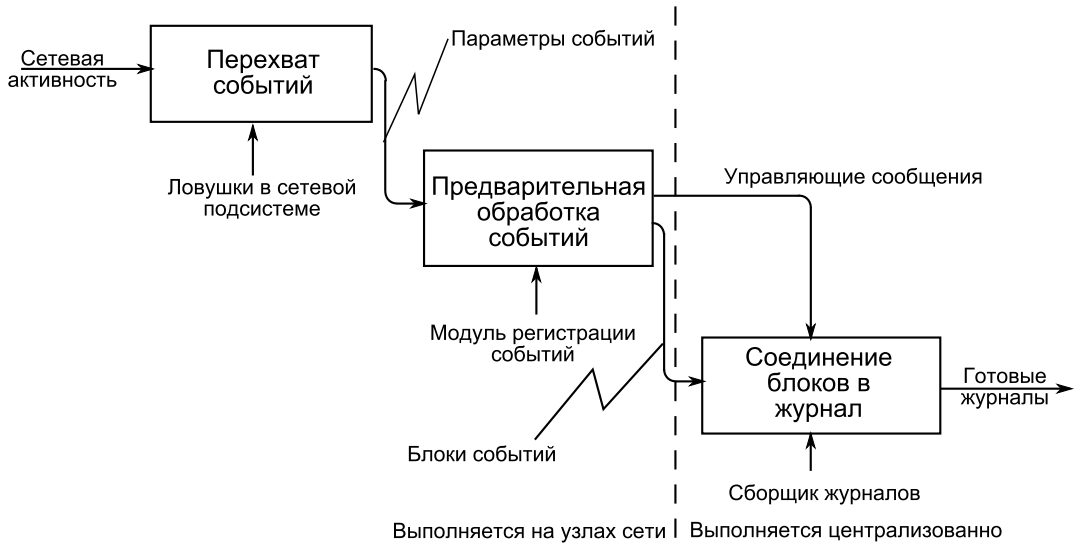

Рис. 3. Функциональная схема разрабатываемого метода

Так как конкретное содержание первых двух этапов зависит от используемой операционной системы, в данной статье подробно рассматривается только третий этап метода, а также его исходные данные. Способ передачи данных от узлов сети «сборщику журналов» в данной статье не уточняется.

3. Алгоритм сборки журналов движения пакетов

Для сборки журналов нужно создать алгоритм, соединяющий полученные события в готовый журнал. Сущности, с которыми будет оперировать алгоритм, и отношения между ними показаны на рис. 4.

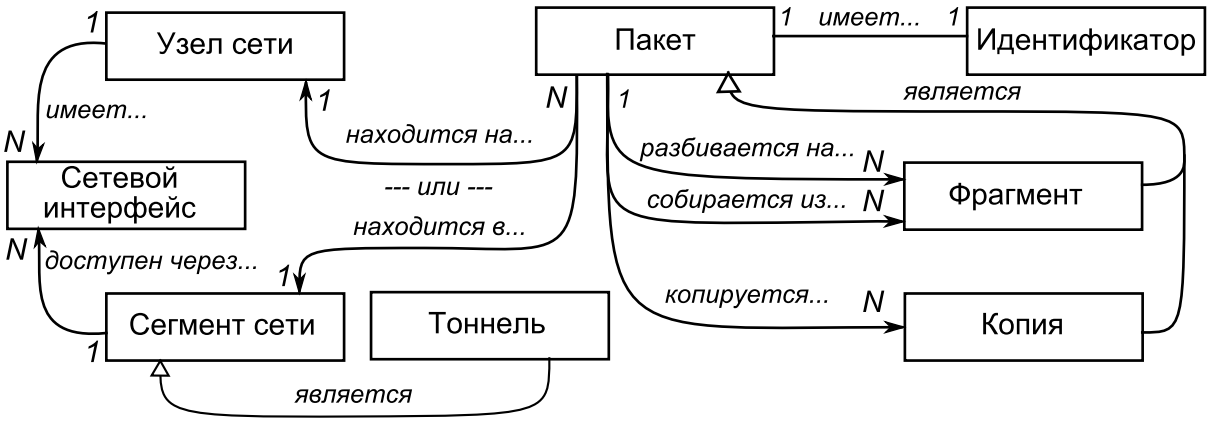

Рис. 4. Инфологическая модель с точки зрения сборщика журналов

Эти отношения реализуются в следующих структурах данных (словарях), используемых алгоритмом построения журнала:

1) *сегменты сети*: ID сегмента сети ↦ набор пакетов, находящихся в этом сегменте;

2) *узлы*: символьное имя узла сети ↦ структура с полями

а) *интерфейсы*: имя интерфейса ↦ ID сегмента сети;

б) *пакеты*: id пакета ↦ блок (все пакеты, находящиеся на узле);

в) *фрагменты*: id фрагмента ↦ блок целого пакета;

г) *копии*: id копии ↦ блок пакета-оригинала;

д) *склейка*: id целого пакета ↦ список блоков фрагментов.

Следующий псевдокод описывает действия, выполняемые сборщиком журналов при получении сообщения от узла сети. Требуемый состав сообщений виден из псевдокода. Заметим, что каждый IP-пакет имеет свой уникальный идентификатор *id*.

```
[от узла H получено сообщение M]
switch (содержание M):
    case уведомление о поступлении пакета id из интерфейса I:
        сегмент сети S := сегменты[узлы[H].интерфейсы[I]]
        блок := извлечь S[id]
        узлы[H].пакеты[id] := блок
    case уведомление о завершении фрагментации пакета id:
        удалить узлы[H].фрагменты[id]
    case уведомление о завершении копирования пакета id:
        удалить узлы[H].копии[id]
    case блок событий E[1]..E[N] для пакета id:
        if E[1] == "формирование нового пакета":
            журнал := новый журнал
            журнал.начало := блок
```

```
        else:
            блок0 := извлечь узлы[H].пакеты[id]
            добавить ссылку блок0 --> блок
        обработать первое событие блока
        обработать последнее событие блока
```

При обработке первого события блока событий выполняется добавление связей между блоками по следующему алгоритму.

```
[обработать первое событие блока]
switch (E[1]):
    case фрагментация пакета id0:
        добавить ссылку узлы[H].фрагменты[id0] --> блок
    case многоадресная рассылка id0:
        добавить ссылку узлы[H].копии[id0] --> блок
    case дефрагментация:
        for each блок0 in узлы[H].склейка[id]:
            добавить ссылку блок0 --> блок
        удалить узлы[H].склейка[id]
    default:
        ничего не делать
```

На основании последнего события в блоке производится обновление структур данных, его обработка производится следующим образом.

```
[обработать последнее событие блока]
switch (E[N]):
    case фрагментация:
        узлы[H].фрагменты[id] := новый служебный блок фрагментации
        добавить ссылку блок --> узлы[H].фрагменты[id]
    case многоадресная рассылка:
        узлы[H].копии[id] := новый служебный блок многоадресной рассылки
        добавить ссылку блок --> узлы[H].копии[id]
    case дефрагментация пакета id2:
        блок1 = новый служебный блок сборки
        добавить ссылку блок --> блок1
        if ключ id2 отсутствует в словаре узлы[H].склейка:
            узлы[H].склейка[id2] := новый список
        добавить блок1 в список узлы[H].склейка[id2]
```

```
case передача в сетевой интерфейс I:
    сегмент сети S = сегменты[узлы[H].интерфейсы[I]]
    S[P] := блок
case доставка по назначению или сброс:
    закрыть журнал пакета
```

Для корректной работы алгоритма сообщения от узлов сети должны поступать на вход сборщика строго в хронологическом порядке.

4. Программная реализация метода

Архитектура реализующего разработанный метод программного комплекса в общем виде представлена на рис. 5.

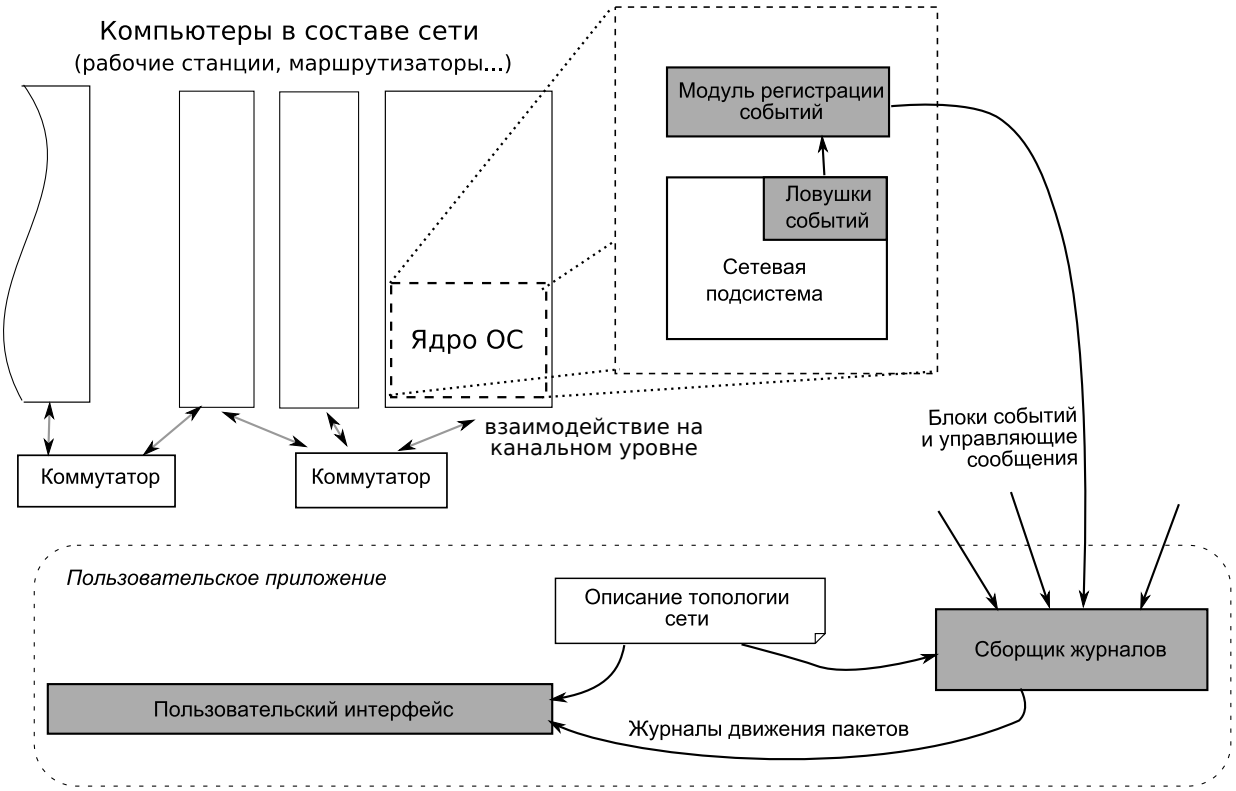

Рис. 5. Архитектура программного комплекса

Названия модулей комплекса соответствуют механизмам, обозначенным на рис. 3. Алгоритм работы сборщика журналов был приведен в разделе 3. Следует отметить, что ловушки событий и модуль регистрации событий тесно взаимодействуют с сетевой подсистемой ОС.

5. Применение метода на практике

Описанный в работе метод отслеживания жизненного цикла IP-пакетов был реализован для среды Netkit [4] — инструмента, позволяющего связать в сеть виртуальные машины на основе ОС Linux. Далее описан опыт с реализованным программным комплексом.

Рассмотрим простую сеть из трех узлов, показанную на рис. 6.

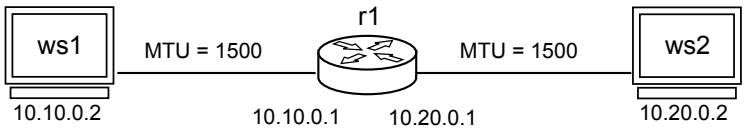

Рис. 6. Топология сети для опыта

Выполним на узле **ws1** команду **ping -c 1 -s 2000 10.20.0.2**, при этом будет создан IP-пакет размером 2028 байт. Основываясь на [1], можно предположить следующий исход опыта: узел **ws1** разбивает пакет на два фрагмента, размер которых не превышает значение MTU (1500 байт), которые будут собраны на узле-получателе **ws2**.

Созданная система показывает отличающийся от ожидаемого журнал (рис. 7). Маршрутизатор **r1** собирает поступившие фрагменты и затем заново разбивает результат на две такие же части перед дальнейшей посылкой. Блок событий, предшествующих дефрагментации на маршрутизаторе **r1**, показан ниже.

```
[ пакет 0C885880 на узле r1 ]
1  <---пакет получен из интерфейса eth0
2  начало обработки сетевым экраном в ловушке PRE_ROUTING
3  пакет помещен в очередь фрагментов для дефрагментации
4  (дефрагментация)
```

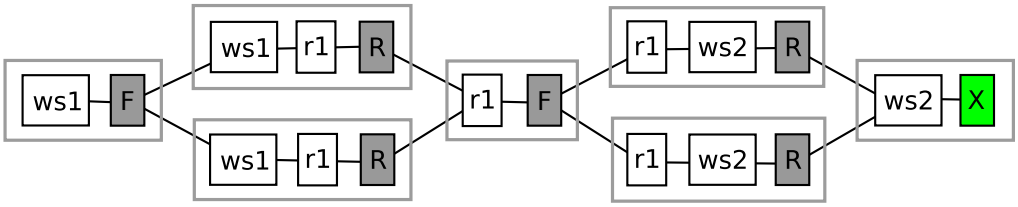

Рис. 7. Журнал пакета. Обозначения служебных блоков:
F — фрагментация, R — дефрагментация, X — доставка

Судя по строкам 5-6, процедура дефрагментации была инициирована после попадания в ловушку сетевого экрана Netfilter, но до выхода из этой ловушки. Изучение исходных кодов ядра Linux объясняет наблюдаемый эффект: принудительная дефрагментация вызывается из функции **ipv4_conntrack_defrag** модуля отслеживания соединений для всех входящих фрагментированных пакетов.

Заключение

В работе был создан метод, позволяющий отследить движение отдельных IP-пакетов по сети и получить полную информацию о событиях их жизненного цикла. Описана архитектура программного комплекса, реализующего разработанный метод для систем на основе ядла Linux и сетевого симулятора Netkit. Приведен опыт с программной реализацией метода, показывающий неочевидную особенность сетевой подсистемы ОС Linux.

Список литературы

1. *Postel J.* Internet Protocol. – RFC 791 (Standard). – 1981.
2. *Deering St.* 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. – Pp. 259–269.
4. *Pizzonia M.*, *Rimondini M.* Easy Emulation of Complex Networks on Inexpensive Hardware // Proc. 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2008). Innsbruck. – 2008. Pp. 7:1–7:10.
5. *Klaus Wehrle.* Linux Network Architecture // Klaus Wehrle, Frank Pahlke, Hartmut Ritter et al. – Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004. – P. 648.

electronic scientific and technical periodical

SCIENCE and EDUCATION

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

IP packet lifecycle tracking method

77-30569/400433

 # 04, April 2012

A.O. Kryuchkov, V.A. Krishchenko

Bauman Moscow State Technical University
kva@bmstu.ru

The paper describes a method to reconstruct the entire lifecycle of an IP packet from its creation to delivery. The method keeps track of events such as fragmentation and reassembly, multicast routing, tunneling and NAT. A software implementation for Linux-based networks is described. The method is applicable to both IPv4 and IPv6.

Keywords: network traffic analysis, IP protocol, IP packets.

References

1. *Postel J.* Internet Protocol. – RFC 791 (Standard). – 1981.
2. *Deering St.* 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. – Pp. 259–269.
4. *Pizzonia M.*, *Rimondini M.* Easy Emulation of Complex Networks on Inexpensive Hardware // Proc. 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2008). Innsbruck. – 2008. Pp. 7:1–7:10.
5. *Klaus Wehrle.* Linux Network Architecture // Klaus Wehrle, Frank Pahlke, Hartmut Ritter et al. – Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004. – P. 648.