Фильтры захвата для сетевых анализаторов (tcpdump, Wireshark, Paketyzer)

1. Фильтры захвата

Анализаторы трафика являются полезным и эффективным инструментом в жизни администратора сети, они позволяют «увидеть» то что на самом деле передается в сети, чем упрощают диагностику разнообразных проблем или же изучение принципов работы тех или иных протоколов и технологий.
Однако в сети зачастую передается достаточно много разнообразных блоков данных, и если заставить вывести на экран все, что проходит через сетевой интерфейс, выделить то, что действительно необходимо, бывает проблематично.
Для решения этой проблемы в анализаторах трафика реализованы фильтры, которые разделены на два типа: фильтры захвата и фильтры отображения. Сегодня пойдет речь о первом типе фильтров – о фильтрах захвата.
Фильтры захвата, это разновидность фильтров, позволяющая ограничить захват кадров только теми, которые необходимы для анализа, уменьшив, таким образом, нагрузку на вычислительные ресурсы компьютера, а также упростив процесс анализа трафика.

2. Синтаксис фильтров захвата

Выражение фильтра захвата состоит из набора специальных примитивов, которые строятся из так называемых классификаторов и идентификаторов объекта (адреса, имена объектов сети, номера портов).

> **Внимание**: все классификаторы регистрозависимы и должны писаться только маленькими буквами.

Давайте разберемся с ними подробней.
Классификаторы могут быть следующих разновидностей:

- **type** – тип объекта
    - **host** – узел (тип по умолчанию, если тип не задан, предполагается что это host)
    - **net** – сеть
    - **port** – порт

Например:
host 192.168.0.1 – захват трафика в котором в качестве адреса (отправителя или получателя) стоит IP 192.168.0.1
net 172.16.0.0/16 – захват трафика в котором в качестве адреса (отправителя или получателя) стоит IP из сети 172.16.0.0/16 (точнее находится в диапазоне от 172.16.0.0 до 172.16.255.255), при этом, так как это всего лишь фильтр поиска совпадающих адресов, совершенно не важно какая настроена маска на интерфейсе, и вас не должно смущать что 172.16.0.0 по маске /16 это номер сети, мы совершенно не знаем какая маска настроена на интерфейсе, и формально, такой адрес узла допустим.
port 80 – захват трафика в котором есть данные принадлежащие порту 80 (udp или tcp)
10.0.0.1 – захват трафика в котором в качестве адреса (отправителя или получателя) стоит IP 10.0.0.1, классификатор host не указан, но он предполагается по умолчанию.

- **dir** – направление по отношению к объекту (direction)
    - **src** – объект является отправителем
    - **dst** – объект является получателем

Например:
src host 192.168.0.1 – захват трафика в котором в качестве адреса отправителя (не получателя) стоит IP 192.168.0.1
dst net 172.16.0.0/16 – захват трафика в котором в качестве адреса получателя (не отправителя) стоит IP из сети 172.16.0.0/16 (точнее находится в диапазоне от 172.16.0.0 до 172.16.255.255).

- **proto** – протокол взаимодействия
    - **ether** – базовая сетевая технология Ethernet, как правило указывает на то что в фильтре используется аппаратный MAC адрес
    - **ip** – протокол IPv4
    - **ip6** – протокол IPv6
    - **arp** – протокол ARP
    - **tcp** – протокол TCP
    - **udp** – протокол UDP
    - если протокол не указан, то считается что должен захватываться весь трафик, совместимый с типом объекта

Например:
src ether host 00:11:22:33:44:55 – захват трафика в котором в качестве MAC адреса отправителя используется 00:11:22:33:44:55.
ip icmp – захват ICMP пакетов.
tcp port 80 – захват трафика в котором есть данные принадлежащие TCP порту 80

Кроме идентификаторов и классификаторов объекта фильтры могут содержать ключевые слова **gateway**, **broadcast**, **multicast**, **less**, **greater** а так же арифметические выражения.

Например:
ip multicast – захват ip пакетов, содержащих адреса из класса D.
less 1000 – захват кадров, у которых размер менее 1000 байт.

Связка нескольких условий может происходить с помощью логических операций:

- «И» – and (&&)
- «ИЛИ» – or (||)
- «НЕ» – not (!) – инверсия значения

При этом приоритет этих операций следующий:

- наивысшим приоритетом обладает операция инверсии
- потом логическое «И»
- наименьшим приоритетом обладает операция «ИЛИ».

Как и в обычных математических выражениях, приоритет можно менять с использованием круглых скобочек (), действия в которых выполняются в первую очередь.

Например:
net 192.168.0.0/24 and tcp port 21 – захват трафика принадлежащего сети (диапазону) 192.168.0.0/24 (или отправитель или получатель) и передающего данные по протоколу TCP и использующего порт 21.
host 192.168.0.1 or host 192.168.0.221 – захват трафика принадлежащего или хосту 192.168.0.1 или хосту 192.168.0.221 (при чем не важно кто отправитель, кто получатель и так же достаточно выполнения одно из двух условий, что бы хотя бы один из этих адресов присутствовал в кадре)
host 192.168.0.1 or host 192.168.0.2 and tcp port 22 – захват или любого трафика принадлежащего хосту 192.168.0.1 или трафика протокола TCP и использующего порт 22 принадлежащего хосту 192.168.0.2.
(host 192.168.0.1 or host 192.168.0.2) and tcp port 22 – захват трафика протокола TCP и использующего порт 22 принадлежащего хосту 192.168.0.1 или хосту 192.168.0.2 (любому из них, или обоим сразу).
(host 192.168.0.1 || host 192.168.0.1) && not tcp port 22 – захват любого трафика кроме трафика протокола TCP и использующего порт 22 принадлежащего хосту 192.168.0.1 или хосту 192.168.0.2 (любому из них, или обоим сразу).

Если в фильтре есть несколько одинаковых повторяющихся классификаторов, то для сокращения записи их можно не писать.

> **Например**:
> **net 192.168.0.0/24 and (tcp port 21 or tcp port 20 or tcp port 25 or tcp port 80 or tcp port 110)**
> можно сократить до
> **net 192.168.0.0/24 and (tcp port 21 or 20 or 25 or 80 or 110)**

> **Внимание**:
> Выражение исключающее пакеты, в которых есть адреса 1.1.1.1 и 1.1.1.2:
> **not (host 1.1.1.1 and host 1.1.1.2)**
> Можно сократить как:
> **not (host 1.1.1.1 and 1.1.1.2)**
> Но не как:
> **not host 1.1.1.1 and 1.1.1.2** – в этом случае будут показаны пакеты в которых нет первого адреса и есть второй.
> И не так
> **not (host 1.1.1.1 or 1.1.1.2)** – в этом случае будут исключены пакеты в которых есть хотя бы один из указанных двух адресов.

Список основных примитивов, которые могут быть использованы для написания фильтров захвата, показан в таблице 2-1.

Таблица 2-1. Список основных примитивов, которые могут быть использованы для написания фильтров захвата.

Примитив	Описание
dst host ip_address	Захватывать кадры, в которых в поле адреса получателя заголовка IPv4/IPv6 содержит заданный адрес узла
src host ip_address	Захватывать кадры, в которых в поле адреса отправителя заголовка IPv4/IPv6 содержит заданный адрес узла
host ip_address	Захватывать кадры, в которых в поле адреса отправителя или получателя заголовка IPv4/IPv6 содержит заданный адрес узла. Эквивалентен фильтру: ether proto ip and host ip_address
ether dst mac_address	Захватывать кадры, в которых в поле адреса получателя заголовка канального уровня содержит заданный MAC адрес узла
ether src mac_address	Захватывать кадры, в которых в поле адреса отправителя заголовка канального уровня содержит заданный MAC адрес узла
ether host mac_address	Захватывать кадры, в которых в поле адреса отправителя или получателя заголовка канального уровня содержит заданный MAC адрес узла
dst net network	Захватывать кадры, в которых в поле адреса получателя заголовка IPv4/IPv6 содержит заданный адрес, принадлежащий диапазону указанной классовой сети
src net network	Захватывать кадры, в которых в поле адреса отправителя заголовка IPv4/IPv6 содержит заданный адрес, принадлежащий диапазону указанной классовой сети

net network	Выбирает все пакеты IPv4/IPv6, содержащие адреса из указанной сети в поле отправителя или получателя
net network mask mask	Захватывать кадры, в которых в поле адреса отправителя или получателя заголовка IPv4/IPv6 содержит заданный адрес, принадлежащий диапазону указанной сети
net network/mask_length	Захватывать кадры, в которых в поле адреса отправителя или получателя заголовка IPv4/IPv6 содержит заданный адрес, принадлежащий диапазону указанной сети
dst port port	Захватывать кадры, в которых в поле порт получателя заголовка UDP или TCP содержит заданный номер порта
src port port	Захватывать кадры, в которых в поле порт отправителя заголовка UDP или TCP содержит заданный номер порта
port port	Захватывать кадры, в которых в поле порт отправителя заголовка UDP или TCP содержит заданный номер порта
less length	Захватывать кадры, размер которых не больше указанного значения
greater length	Захватывать кадры, размер которых не меньше указанного значения
ip proto protocol	Захватывать кадры, в которых в поле «Protocol» заголовка IPv4, содержится идентификатор указанного протокола. При этом можно указывать не только численные значения протоколов, но и их стандартные имена (icmp, igmp, igrp, pim, ah, esp, vrrp, udp, tcp и другие). Однако следует учитывать, что tcp, udp и icmp используются также в качестве ключевых слов, поэтому перед этими символьными идентификаторами следует помешать символ обратного слэша («\»)
ip6 proto protocol	Захватывать кадры, в которых в поле «Protocol» заголовка IPv4, содержится идентификатор указанного протокола. При этом можно указывать не только численные значения протоколов, но и их стандартные имена (icmp6, igmp, igrp, pim, ah, esp, vrrp, udp, tcp и другие). Однако следует учитывать, что tcp, udp и icmp6 используются также в качестве ключевых слов, поэтому перед этими символьными идентификаторами следует помешать символ обратного слэша («\»)
ether broadcast	Захватывать все широковещательные кадры Ethernet. Ключевое слово ether может быть опущено
ip broadcast	Захватывать кадры, содержащие широковещательные адреса в заголовке пакета IPv4. При этом для определения, является ли адрес широковещательным, используется маски подсети для интерфейса, который используется для захвата пакетов. Так же захватывает пакеты, отправленные на ограниченный широковещательный адрес
ether multicast	Захватывать все групповые кадры Ethernet. Ключевое слово ether может быть опущено
ip multicast	Захватывать кадры, содержащие групповые адреса в заголовке пакета IPv4
ip6 multicast	Захватывать кадры, содержащие групповые адреса в заголовке пакета IPv6
ether proto protocol_type	Захватывать кадры Ethernet с заданным типом протокола. Протокол может быть указан по номеру или имени (ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl, moprc, iso, stp, ipx, netbeui)
ip, ip6, arp, rarp, atalk, aarp, decnet, iso, stp, ipx, netbeui, tcp, udp, icmp	Захватывать кадры передающие данные указанного протокола. Используются в качестве сокращения для: ether proto protocol

vlan [vlan_id]	Захватывать кадры соответствующе стандарту IEEE 802.1Q. Если при этом указан номер vlan_id, то захватываются только кадры принадлежащие указанному VLAN

3. Расширенные примеры фильтров захвата

Кроме простых указаний адресов и протоколов в фильтрах захвата можно использовать и более сложные конструкции, позволяющие производить более тонкий анализ заголовков. Для этого используются выражения возвращающее логическое значение следующего формата:

> **expression** операция **expression**

В котором в качестве **expression** могут быть константы, результаты арифметических (+, -, *, /) или двоичных побитовых операций (& — «И», | — «ИЛИ», << — сдвиг влево, >> — сдвиг вправо), оператор длинны **offset**, данные или поля заголовков кадра. В качестве операции могут быть применены символы «>» (больше), «<» (меньше), «>=» (больше равно), «<=» (меньше равно), «=» (равно), «!=» (не равно). Таким образом, можно выполнять проверку на совпадение или не совпадение определенных полей или байт кадра с необходимыми значениями, сравнивать разные поля заголовков между собой, а также выполнять над ними некоторые арифметические и логические операции и сравнивать результаты этих операций с определенными значениями.
Простейшим примером использования расширенного фильтра будет «**5 = 3+1**», где «**5**» и «**3+1**» — expression, а «=» — операция. В результате вычисления этой строки будет возвращено логическое значение, в данном случае **false**.

Для получения данных или заголовков кадра используется примитив **proto**[*offset*:*size*].

> **Внимание**: квадратные скобочки в данном случае элемент синтаксиса, а не признак необязательного поля.

Параметр **proto** содержит название протокола, из заголовка которого необходимо выбрать определенные данные (ether, fddi, tr, wlan, ppp, slip, link, ip, arp, rarp, tcp, udp, icmp, ip6 и другие).
Параметр **offset** указывает на смещение в байтах относительно начала заголовка указанного протокола, нумерация байт начинается с нуля: **ip[0]** – первый байт от начала IP пакета, **tcp[1]** – второй байт от начала TCP сегмента, **ether[3]** – четвертый байт от начала Ethernet кадра.
Параметр **size** указывает на количество байт, которое необходимо взять, начиная с байта, указанного в смещении (**offset**), поле **size** является не обязательным, и если оно отсутствует, то считается что необходимо взять 1 байт: **ip[2:2]** – третий и четвертый байты от начала IP пакета, **tcp[4]** – пятый байт от начала TCP сегмента, **ether[6-6]** – байты с седьмого по двенадцатый, от начала Ethernet кадра.

Если в поле **offset** установить отрицательное значение, то будут выбраны байты предыдущего заголовка, идущие до заголовка протокола, указанного в параметре **proto**. Но при этом будет обязательно требоваться наличие в кадре заголовка протокола, указанного в примитиве proto. Таким образом, фильтры **ether[11]=0x37** (взять 12-й байт кадра Ethernet и сравнить его со значением 0x37) и **ip[-3] = 0x37** (взять 3-й байт с конца заголовка, идущего перед заголовком IP, и сравнить его со значением 0x37) не являются идентичными. В первом будут пропускаться все кадры, в которых MAC адрес отправителя заканчивается на 37, а во втором так же будет требоваться наличие протокола IP, а кадры не содержащие IP протокол, например, ARP кадры, захватываться не будут.

> **Например**:
> Выражения **ip[1:1]** и **ip[1]** приведут к одному и тому же результату – будет выбрано значение второго байта заголовка IPv4
> Выражение **tcp[8:2]** выберет девятый и десятый байты (поле Source Port) заголовка TCP.
> Выражение **ip[-3] = 0x37** выберет все IPv4 пакеты, MAC адрес отправителя которых заканчивается на «0x37».

Следует учитывать, что при выборе данных с помощью конструкции **proto** [*offset*:*size*] для протоколов TCP и UDP, учитывается фрагментация IP пакетов. В результате **tcp[0]** всегда будет означать первый байт заголовка TCP, и никогда не приведет к выбору первого байта данных пакетов, передающих не первый фрагмент из цепочки фрагментов.
Для некоторых протоколов определенные поля и значения смещений могут задаваться не только числами, но и именами. Например, для протокола ICMP поддерживается параметр **icmptype**, который может принимать значения **icmp-echoreply**, **icmp-unreach**, **icmp-sourcequench**, **icmp-redirect**, **icmp-echo**, **icmp-routeradvert**, **icmp-routersolicit**, **icmp-timxceed**, **icmp-paramprob**, **icmp-tstamp**, **icmp-tstam-preply**, **icmp-ireq**, **icmp-ireqreply**, **icmp-maskreq**, **icmp-maskreply**. Для анализа флагов TCP можно использовать параметр **tcpflags** идентификаторы **tcp-fin**, **tcp-syn**, **tcp-rst**, **tcp-push**, **tcp-ack** и **tcp-urg**.

> **Например**:
> Выражение **tcp[tcpflags]&(tcp-syn|tcp-fin) != 0** выберет все кадры, содержащие TCP сегменты в которых открывается или завершается сессия.
> Выражение i**cmp[icmptype]!=icmp-echo and icmp[icmptype]!=icmp-echoreply** выберет все кадры, содержащие ICMP протокол, кроме эхо запросов и эхо ответов.

Могут быть ситуации, в которых необходимо анализировать только часть бит, определенного байта. Для решения этих задач используется битовая операция «И» (&). С ее помощью можно сохранить только определенные биты байта, а остальные обнулить.
Например, нам необходимо выделить только те кадры, которые передаются на канальном уровне широковещательными или групповыми кадрами. Мы знаем, что определить тип MAC адреса можно по его старшему байту:

Тип адреса	Значение старшего байта в 16-й системе	Значение старшего байта в 2-й системе
Направленные\Unicast	00	00000000
Групповые\Multicast	01	00000001
Административно назначенные\Admin ID	01	00000010
Широковещательные\Broadcast	FF	11111111

Исходя из этой информации, можно сделать вывод о том, то в широковещательных или групповых адресах младший бит старшего байта адреса равен единице, а в остальных – нулю. Если мы возьмем старший байта адреса, обнулим все его биты кроме самого младшего, и при этом значение байта станет равным единице, то этот адрес был или широковещательным, или групповым, если значение байта станет равным нулю, то этот адрес были или направленным, или административно заданным. В результате для выполнения данной проверки необходимо использоваться следующее выражение: **ether[0]&1 = 1**, где **ether[0]** – получает значение первого байта Ethernet заголовка, а **&1** — битовая операция логическое «И», обнуляющая все биты этого байта, кроме младшего, «**= 1**» — проверка результата на совпадение с единицей.
Разберем еще один пример более подробно.
Нам нужно получить содержимое поля Type Of Service (ToS) заголовка IPv4. Для этого обратившись к RFC-791 мы увидим, что это поле является однобайтовым полем, и вторым байтом заголовка:

```
3.1.  Internet Header Format
  A summary of the contents of the internet header follows:
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |Version|  IHL  |Type of Service|          Total Length         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         Identification        |Flags|      Fragment Offset    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Time to Live |    Protocol   |         Header Checksum       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Source Address                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Destination Address                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Options                    |    Padding    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Для того что бы получить его значение нам необходимо воспользоваться следующим примитивом:
ip[1:1] – получить один байт заголовка IP начиная с байта номер 1 (нумерация байт начинается с нуля).
Теперь мы можем строить фильтры, основываясь на содержимом этого поля.
Если мы хотим, чтобы отображались все кадры, содержащие заголовок IPv4, в котором поле ToS равно нулю необходимо написать следующее: **ip[1:1] = 0**.

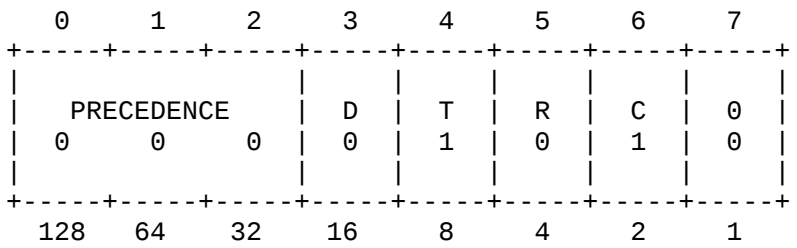
Если мы хотим что бы отображались все кадры, содержащие заголовок IPv4, в котором поле ToS не равно нулю необходимо написать следующее: **ip[1:1] != 0**.
Но можно пойти дальше, согласно RFC-791 поле ToS является составным полем и имеет следующую структуру:

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |     |     |     |     |     |
|   PRECEDENCE    |  D  |  T  |  R  |  0  |  0  |
|                 |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Bits 0-2:  Precedence.
Bit    3:  0 = Normal Delay,      1 = Low Delay.
Bits   4:  0 = Normal Throughput, 1 = High Throughput.
Bits   5:  0 = Normal Relibility, 1 = High Relibility.
Bit  6-7:  Reserved for Future Use.
```

Первые три бита – предпочтительность, четвертый описывает требования к задержкам, пятый описывает требования к пропускной способности, шестой описывает требования к надежности линии связи, седьмой и восьмой – зарезервированы для будущего использования.
Если обратиться к более новым стандартам (RFC1349), значение седьмого бита уже определено – требования по цене – «Cost» (денежный эквивалент).
И вот, допустим, мы хотим определить есть ли в сети кадры, в которых в IPv4 заголовке установлен седьмой бит поля ToS. Как это сделать? Для решения этой задачи нам понадобиться вспомнить (или выучить:D) двоичную систему исчисления. В байте каждый бит имеет свой вес, который начинается с единицы, и увеличивается, справа налево каждый раз умножаясь на два.

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |     |     |     |     |     |
|   PRECEDENCE    |  D  |  T  |  R  |  C  |  0  |
|  0     0     0  |  0  |  0  |  0  |  1  |  0  |
|                 |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
  128   64    32    16     8     4     2     1
```

Получается, что вес интересующего нас бита равен 2.
Что если мы сравним значение поля ToS с двойкой?
ip[1:1] = 2
Получим ли мы ответ на вопрос, присутствует ли в этом заголовке интересующий нас бит? С одной стороны, да, но с другой стороны и нет.
Например, если у нас в поле ToS есть кроме бита «Cost» еще и другие биты, установленные в единицу? Допустим это будет бит отвечающий за требования к пропускной способности – «Throughput».



В результате значение этого байта будет уже не 2, а 10, и простым сравнением нельзя получить ответ на вопрос, установлен ли определенный бит в интересующем нас поле.
Что же нам мешает получить интересующий нас ответ? Нам мешает значение других, возможно также установленных в единицу битов. Соответственно надо от них избавиться.
Для решения этой задачи воспользуемся операцией побитового логического «И» (иногда называется логическое умножение), обозначается символом «**&**». Как известно, в логической операции «И» на выходе будет только тогда единица, когда и первый операнд и второй равны единице. Соответственно если мы произведем побитовое умножение значения поля ToS на специальную маску, в которой будет установлен в единицу только тот бит, который находится на позиции интересующего нас бита в поле ToS, то мы исключим из результата все остальные биты:

```
Поле ToS :00001010=10
Маска    :00000010=2
Результат:00000010=2
```

Каким бы ни было значение остальных бит, при накладывании этой маски в результат попадет только значение интересующего нас поля. Даже если мы установим все биты в единицу, это не повлияет на результат:

```
Поле ToS :11111111=255
Маска    :00000010=2
Результат:00000010=2
```

И только лишь, если интересующий нас бит, равен нулю, в результате наложения маски будет так же ноль.

```
Поле ToS :11111101=253
Маска    :00000010=1
Результат:00000000=0
```

Таким образом, если интересующий нас бит равен единице, в результате наложения маски мы получим вес этого бита, если он равен нулю, то мы получим ноль.
Исходя из этого, для решения этой задачи нам необходимо применить следующий фильтр:
ip[1:1] & 2 = 2

Он будет брать значение второго байта, накладывать на него маску «вырезающую» значение определенного бита и сравнивать результат с весом этого бита.

Можно привести еще один пример на основании анализа поля Type Of Service заголовка IP: нам нужно увидеть все кадры, в которых в заголовке IPv4 в поле ToS значение битов Precedence (предпочтительность) не равна нулю. Для этого применим маску, в который единичками выделим те биты, которые отвечают за Precedence:

```
Поле ToS :10111101=189
Маска    :11100000=224
Результат:10100000=160
```

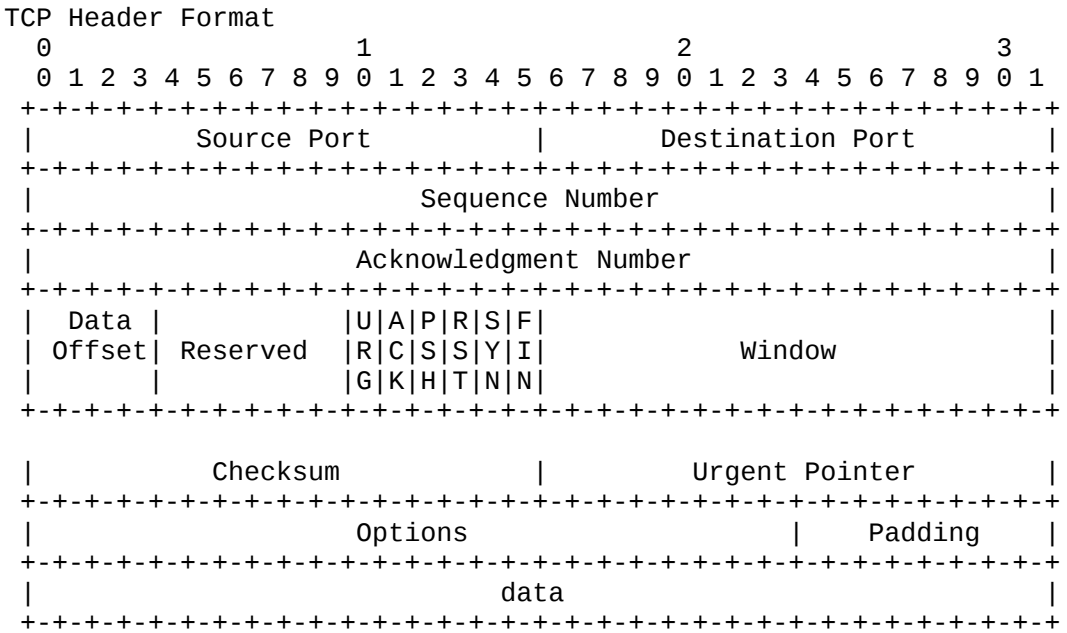
Результат не равен нулю, и это говорит о том, что поле Precedence так же не равно нулю.

```
Поле ToS :00011111=31
Маска    :11100000=224
Результат:00000000=0
```

Результат равен нулю, и это говорит о том, что поле Precedence так же равно нулю.

В результате, проверка на ненулевое значение поля ToS в заголовке IPv4 будет выглядеть следующим образом:
ip[1:1] & 224 != 0
или же тоже самое, но используя шестнадцатеричный вариант:
ip[1:1] & 0xe0 != 0

Рассмотрим пример с другим протоколом. Возьмем протокол TCP.
Например, нам нужно захватить все кадры, которые передают TCP сегменты с опциями. Для того что бы понять, что нужно искать и где обратимся к RFC-793.



Для определения есть ли в сегменте опции, используется поле «Data Offset», оно показывается длину заголовка в четырехбайтовых словах. Минимальная длина заголовка TCP сегмента – 20 байт, а это 5 четырехбайтовых слов. Соответственно, если в заголовке TCP сегмента есть опции, то значение этого поля будет больше 5.
Для того, чтобы получить значение этого поля необходимо воспользоваться примитивом **tcp[12:1]**. Правда, учитывая то что минимальный кусок, который мы можем взять это один байт, а нам нужно всего 4 бита, придется немного подумать.
Применив примитив **tcp[12:1]** мы получили следующий кусок заголовка:

```
+-+-+-+-+-+-+-+-
|  Data |
| Offset| Reserved
|       |
+-+-+-+-+-+-+-+-
```

Если бы поле «Data Offset» было в младшей части байта, то число «5» в двоичном представлении выглядело бы следующим образом:

```
128   64    32    16     8     4     2     1
 0     0     0     0     0     1     0     1 = 5
```

Но интересующие нас биты находятся не в левой, младшей, а в правой, старшей его части, поэтому, для получения десятичного эквивалента переносим их в правую часть байта:

```
128   64    32    16     8     4     2     1
  0    1     0     1     0     0     0     0 = 80 (0x50 в
шестнадцатеричном виде)
```

Для выделения старших бит необходимо наложить маску:

```
Поле Data Offset :01010000=80
Маска            :11110000=240
Результат        :01010000=80
```

Если в заголовке все же есть опции, то значение «Data Offset» будет больше 5. Например, если в заголовке будет одна восьмибайтовая опция, то значение этого поля будет 7 (5 четырехбайтовых слов фиксированной части заголовка, и 2 четырехбайтовых слова опции):

```
128   64    32    16     8     4     2     1
 0     0     0     0     0     1     1     1 = 7
```

Перенеся соответствующие биты в старшую часть получаем:

```
128   64    32    16     8     4     2     1
  0    1     1     1     0     0     0     0 = 112 (0x70 в
шестнадцатеричном виде)
```

Выделяем старшие биты накладывая маску:

```
Поле Data Offset :01110000=112
Маска            :11110000=240
Результат        :01110000=112
```

Таким образом, получается, что если в результате получается значение больше 80, то в TCP заголовке есть опции. В принципе тут маску можно было и не накладывать, так как лишние биты все равно резервные, и всегда должны быть равны нулю, но мало ли, что может измениться, и чтобы не переписывать фильтр, если вдруг стандарт измениться, мы лучше их обрежем маской.
Результирующий фильтр, который покажет те TCP сегменты, в которых длина TCP заголовка больше 5 четырехбайтовых слов, получился следующий:
tcp[12:1] & 240 != 80
или
tcp[12:1] & 240 > 80
или
tcp[12:1] & 0xf0 > 80
Также давайте рассмотрим возможность работы с TCP флагами. Их можно выделять таким же методом с помощью маски, но также можно использовать символьные классификаторы, которые приводились выше.
Например, для того что бы захватить кадры содержащие сегменты с флагами SYN или FIN, необходимо написать следующий фильтр:
tcp[tcpflags] & (tcp-syn|tcp-fin) != 0
Я думаю он вполне читаем и не требует особых пояснений.
Реализация подобной задачи через биты и маски привела бы к такому формату фильтра:
tcp[13:1] & 2 != 0 or tcp[13:1] & 1 != 0
В качестве закрепления понимания темы попробуйте самостоятельно разобраться с тем, как этот вариант фильтра будет работать.

> **Например**:
> Выражение **ether[0]&1 != 0** выберет все широковещательные кадры.
> Выражение **ether[0] & 1 = 0 and ip [16]>= 244** выберет все широковещательные или групповые IP пакеты, в которых на канальном уровне не используется широковещательный или групповой MAC адрес.
> Выражение **ip[0]&0xf = 5** выберет все IP пакеты, в которых нет опций.
> Выражение **ip[6:2]&0x1fff = 0** выберет все не фрагментированные IP пакеты, и первые фрагменты фрагментированных пакетов.
> Выражение **ip[-3]&0xff = 0x37** выберет все IP пакеты, MAC адрес отправителя которых заканчивается на «0x37».

Еще одним интересным набором битовых операций является операции битового сдвига. Эти операции обозначаются символами «пара стрелочек»: «<<» — сдвиг влево и «>>»- сдвиг вправо.
Как они работают?

Возьмем произвольный байт, для простоты возьмем единицу, и запишем ее значение в двоичной системе исчисления:

```
128   64    32    16     8     4     2     1
  0    0     0     0     0     0     0     1 = 1
```

Теперь произведем операцию битового сдвига влево, смещая значения всех бит на одну позицию, и поставляя ноль в младший освободившийся разряд:

```
128   64    32    16     8     4     2     1
 0     0     0     0     0     0     1     <<
```

```
128   64    32    16     8     4     2     1
 0     0     0     0     0     0     1     0 = 2
```

В результате значение байта стала равно двойке, то есть увеличилось в два раза. И еще раз применим эту операцию:

```
128   64    32    16     8     4     2     1
 0     0     0     0     0     1     0     <<
```

```
128   64    32    16     8     4     2     1
 0     0     0     0     0     1     0     0 = 4
```

В результате значение байта стала равно четверке, то есть опять увеличилось в два раза. Таким образом, можно сделать вывод, что операция битового сдвига влево, эквивалентна умножению значения байта на два (так как мы работаем с двоичной системой исчисления). Давайте проверим действительность этого правила на более сложном числе, например, 100. Запишем его в двоичном виде:

```
128   64    32    16     8     4     2     1
  0    1     1     0     0     1     0     0 = 100
```

А теперь произведем операцию сдвига влево:

```
128   64    32    16     8     4     2     1
 1     1     0     0     1     0     0     <<
```

```
128   64    32    16     8     4     2     1
 1     1     0     0     1     0     0     0 = 200
```

В результате значение байта стала равно 200 – увеличилось в два раза.

Соответственно можно сразу сделать вывод о том, что битовый сдвиг вправо эквивалентен делению числа на два.
Например, число 240:

```
128   64    32    16     8     4     2     1
  1    1     1     1     0     0     0     0 = 240
```

Произведем операцию сдвига вправо:

```
128   64    32    16     8     4     2     1
 >>    1     1     1     1     0     0     0
```

```
128   64    32    16     8     4     2     1
 0     1     1     1     1     0     0     0 = 120
```

Значение байта стала равно 120 – уменьшилось в два раза.
Как эту операцию мы можем использовать? Давайте вспомним, что поле IHL (Internet Header Length) заголовка IP указывает длину заголовка не в байтах, а в четырехбайтовых словах, и для того что бы проверить, содержит ли пакет опции мы применяли следующую операцию:
ip[0]&0xf = 5
То есть сравнивали не с реальным значением, а со значением, разделенным на четыре (20 байт это 5 четырехбайтовых слов). Если по каким-то причинам удобней работать с длиной заголовка в байтах (например, если это значение надо впоследствии вычесть из общей длины пакета), то его необходимо умножить на 4. Для того что бы умножить число на 4, его нужно дважды умножить на два, то есть провести операцию битового сдвига влево дважды, после чего сравнить с необходимой длинной IP заголовка в байтах:
ip[0]<<2 = 20

Ну и конечно же все это можно объединять в составные наборы правил:
(icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply) or (udp and udp port not 67 and ip[16] < 224) or (tcp[0:2]<1024 and tcp[2:2]<1024)
С этим фильтром программа будет захватывать только те кадры, которые подходят под одно из трех описаний:

- Содержат ICMP сообщения, кроме echo и echoreply (использующиеся утилитой ping)
- Передают UDP дейтаграммы, кроме тех, что используют в качестве порта отправителя или порта получателя порт 67 и кроме тех, которые передаются на групповые адреса и на ограниченный широковещательный адрес
- Передают TCP сегменты, в которых и порт отправителя, и порт получателя находятся в диапазоне «Хорошо известных портов»

4. Задание для самопроверки :D

Для закрепления работы со сложными фильтрами захвата попробуйте понять, что описывает и как работает этот фильтр:
tcp port 80 and (ip[2:2] — ip[0]&0xf<<2 — tcp[12]&0xf0>>2 != 0)

> **Примечание**:
> Это один из стандартных примеров, и при необходимости вы легко сможете проверить себя, попользовавшись поиском в интернет, но все же попробуйте приложить усилия и самостоятельно с ним разобраться.