Несколько слов о Path MTU Discovery Black Hole

Автор: Попов Александр (popov2al<at>gmail.com)

Вместо вступления
Однажды для каждого настоящего системного администратора (или исполняющего обязанности такового) наступает момент истины. Ему выпадает судьба настроить маршрутизатор на компьютере с установленной ОС GNU/Linux. Те, кто это уже прошел, знают, что ничего сложного в этом нет и можно уложиться в пару команд. И вот наш админ находит эти команды, вбивает их в консоль и гордо идет к пользователям сказать, что уже все работает. Но не тут-то было – пользователи говорят что их любимые сайты не открываются. После траты некоторой части своей жизни на выяснение подробностей обнаруживается, что большая часть сайтов ведет себя следующим образом:
1. При открытии страницы загружается заголовок и больше ничего;
2. В таком состоянии страница висит неопределенно долгое время;
3. Строка статуса браузера все это время показывает что загружает страницу;
4. Пинги и трассировка до данного сайта проходят нормально;
5. Соединение по telnet на 80 порт тоже проходит нормально.
Обескураженный админ звонит в техподдержку провайдера, но там от него быстро избавляются, советуя попробовать настроить маршрутизатор на ОС Windows, а если уж и там не работает тогда… – купить аппаратный маршрутизатор .
Я думаю, эта ситуация знакома многим. Некоторые в нее попадали сами, у кого-то с ней сталкивались знакомые, а кто-то встречал таких админов на форумах и прочих конференциях. Итак : если у Вас Такая Ситуация, то – Поздравляю! Вы столкнулись с **Path MTU Discovery Black Hole**. Данная статья посвящается тому, отчего это бывает, и как решить эту проблему.

Термины, необходимые для понимания содержимого статьи:
MTU (Maximum Transmission Unit) – этот термин используется для определения максимального размера пакета (в байтах), который может быть передан на канальном уровне сетевой модели OSI. Для Ethernet это 1500 байт. Если приходит пакет большего размера (например по Token Ring), то данные пересобираются в пакеты размером не более MTU(т е не более 1500 байт). Операция пересборки пакетов под другой MTU называется фрагментацией(fragmentation) и считается затратной для маршрутизатора.
PMTU (Path MTU) – данный параметр обозначает наименьший MTU любого канала данных, находящегося между источником и приемником.
PMTU discovery – технология определения PMTU разработанная для уменьшения нагрузки на маршрутизаторы. Описана в RFC 1191(http://tools.ietf.org/html/rfc1191) в 1988 году. Суть технологии заключается в том, что при соединении двух хостов устанавливается параметр DF (don't fragment, не фрагментировать), который запрещает фрагментацию пакетов. Это приводит к тому, что узел, значение MTU которого меньше размера пакета, отклоняет передачу пакета и отправляет сообщение ICMP типа Destination is unreachable (Хост недоступен). К сообщению об ошибке прилагается значение MTU узла. Хост-отправитель уменьшает размер пакета и отсылает его заново. Такая операция происходит до тех пор, пока пакет не будет достаточно мал, чтобы дойти до хоста-получателя без фрагментации.
MSS(Maximum Segment Size) – максимальный размер сегмента, т.е. самая большая порция данных, которую TCP пошлет на удаленный другой конец соединения. Рассчитывается по следующей формуле: [MTU интерфейса] – [Размер IP заголовка(20 байт)] – [Размер TCP заголовка(20 байт)]. Итого обычно это 1460 байт. Когда соединение устанавливается, каждая сторона может объявить свой MSS. Выбирается наименьшее значение. Подробнее смотреть здесь – http://www.cyberguru.ru/networks/protocols…tion-page3.html
Флаг DF(Don't fragment) – Бит в поле флагов заголовка IP пакета, который будучи установленным в единицу сообщает о том,

что данный пакет запрещено фрагментировать. Если пакет с таким флагом больше, чем MTU следующей пересылки, то этот пакет будет отброшен, а отправителю посылается ICMP ошибка “фрагментация необходима, однако установлен бит не фрагментировать” (fragmentation needed but don’t fragment bit set).

Для начала посмотрим, что происходит в сети при открытии страницы. Для этого я создал тестовую сеть, изображенную на рис.1

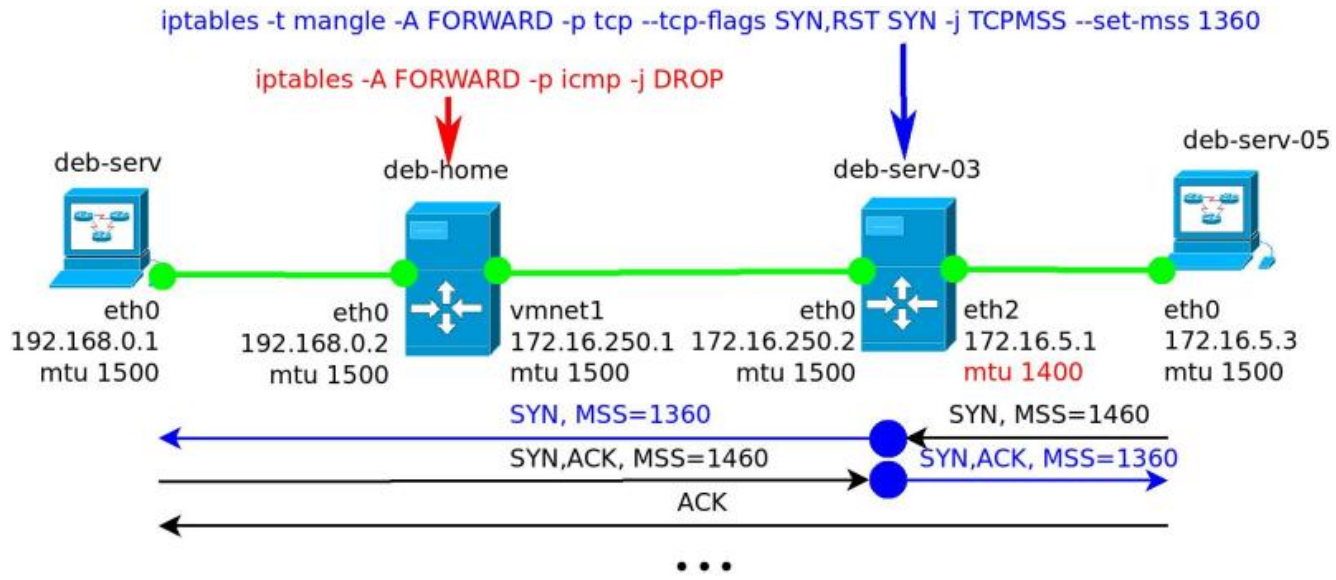

Рис. 1. Схема тестовой сети.

Это упрощенный вариант глобальной сети. Компьютер с именем deb-serv-03 представляет собой наш маршрутизатор на Linux; deb-serv-05 – клиент в локальной сети(); deb-home – маршрутизатор, расположенный у провайдера; deb-serv – Веб-сервер в Интернете с которым мы хотим обмениваться данными(получаем с www.site.local страничку размером в 5,9Кб). Конечно, в реальности цепочка гораздо больше, но для показательного примера этого хватит. Все компьютеры данной сети работают под управлением Debian GNU/Linux 5.0 Lenny. В разных точках сети я контролирую ситуацию с помощью программы tcpdump.

Внимание – на интерфейсе eth2 компьютера deb-serv-03 размер MTU уменьшен до 1400 байт.

Изучаем, как пойдут пакеты при получении странички с веб-сервера. Смотрим на вывод TCPDUMP#1 (на eth0 deb-serv):

Код: Выделить всё

```
1     IP 172.16.5.3.48547 > 192.168.0.1.80: Flags [S], seq 2947128725, win 5840, options [mss 1460...], length 0
2     IP 192.168.0.1.80 > 172.16.5.3.48547: Flags [S.], seq 757312786, ack 2947128726, win 5792, options [mss 1460...], length 0
3     IP 172.16.5.3.48547 > 192.168.0.1.80: Flags [.], ack 1, win 1460, options [...], length 0
4     IP 172.16.5.3.48547 > 192.168.0.1.80: Flags [P.], seq 1:118, ack 1, win 1460, options [...], length 117
5     IP 192.168.0.1.80 > 172.16.5.3.48547: Flags [.], ack 118, win 181, options [...], length 0
6     IP 192.168.0.1.80 > 172.16.5.3.48547: Flags [.], seq 1:2897, ack 118, win 181, options [...], length 2896
7      IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3 unreachable - need to frag (mtu 1400), length 556
8     IP 192.168.0.1.80 > 172.16.5.3.48547: Flags [.], seq 1:1349, ack 118, win 181, options [...], length 1348
9     IP 192.168.0.1.80 > 172.16.5.3.48547: Flags [.], seq 1349:2697, ack 118, win 181, options [...], length 1348
10      IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3 unreachable - need to frag (mtu 1400), length 556
```

Я привожу только первые 10 пакетов и убрал неважные для данного примера опции. Разбираем:
1. В строках с 1-ой по 3-ю мы видим установку tcp соединения. Стороны обмениваются пакетами SYN, SYN-ACK, ACK. Здесь стоит обратить внимание на поле опций, а именно на параметр MSS, которым обмениваются стороны. С обеих сторон это 1460 байт. Значит максимальный размер пакетов, которые стороны будут посылать друг другу, составит 1460(MSS)+20(TCP Заголовок)+20(IP Заголовок)=1500 байт.
2. В строке 4 отправка запроса на получение веб страницы от

deb-serv-05. В строке 5 подтверждение получения данного пакета.
3. В строке 6 мы видим отправку ответа на запрос (т.е. отправку куска веб-страницы). Я не знаю почему, но на данном интерфейсе tcpdump видит один пакет размером в 2948 байт, в то время как в сеть уйдут 2 пакета размером 1500 и 1452 байта соответственно. Если посмотреть более подробный вывод tcpdump, то увидим, что на данном пакете(точнее пакетах) стоит флаг DF:

Код: Выделить всё

```
IP (tos 0x0, ttl 64, id 5177, offset 0, flags [DF], proto TCP (6), length 2948)
192.168.0.1.80 > 172.16.5.3.48547: Flags [.], seq 1:2897, ack 118, win 181, options [nop,nop,TS val 86620459 ecr 4922429], length 2896
```

4. Когда эти пакеты с данными доходят до deb-serv-03 они отбрасываются, так как не могут пройти по соединению с MTU 1400 и не могут быть фрагментированы(флаг DF), а в ответ генерируется сообщение ICMP тип 3 код 4: ICMP 172.16.5.3 unreachable – need to frag (mtu 1400), которое мы видим в строке 7 (в строке 10 приходит сообщение для 2-го пакета). В этом сообщении передается нужный MTU.
5. В строках 8 и 9 мы наблюдаем как deb-serv, получив MTU=1400, отправляет тот же самый кусок веб страницы в пакетах размером 1400 байт. Данные пакеты доходят до deb-serv-05, где генерируется подтверждение, и так повторяется до тех пор, пока вся страница не будет передана. Размер всех последующих пакетов будет не больше 1400 байт.
На этом примере демонстрируется процедура определения Транспортного MTU (PMTU), описанная в RCF1911 (http://tools.ietf.org/html/rfc1191). Я представил ее в упрощенном виде на рис 2.

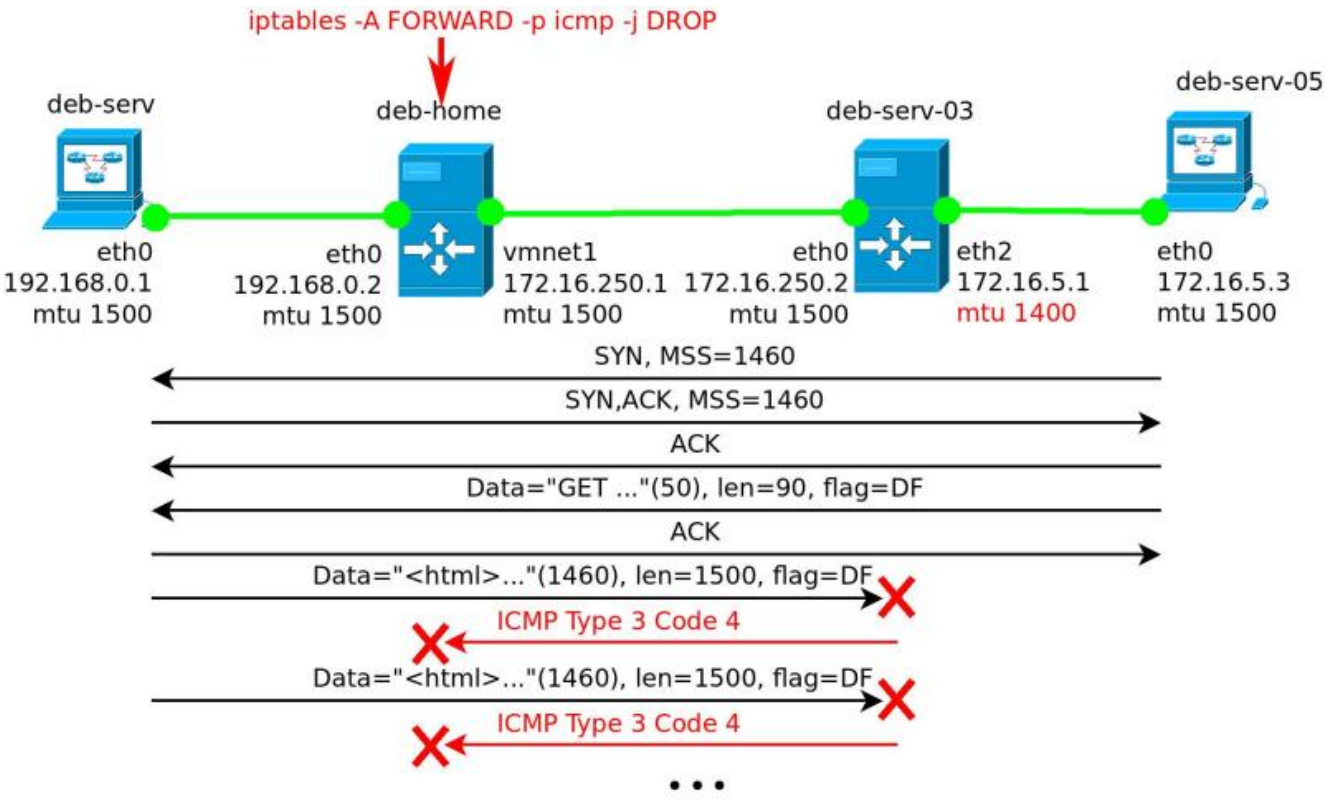

Рис. 2. Процедура определения PMTU.

А теперь представим, что к провайдеру пришел новый специалист и решил (в целях защиты от icmp флуда) запретить пересылку icmp пакетов через deb-home, который теперь в его ведении. Смотрим что получается:
Вывод TCPDUMP#1 (на eth0 deb-serv):

Код: Выделить всё

```
1     IP 172.16.5.3.57925 > 192.168.0.1.80: Flags [S], seq 1723325723, win 5840, options [mss 1460...], length 0
2     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [S.], seq 2482933888, ack 1723325724, win 5792, options [mss 1460...], length 0
3     IP 172.16.5.3.57925 > 192.168.0.1.80: Flags [.], ack 1, win 1460, options [...], length 0
4     IP 172.16.5.3.57925 > 192.168.0.1.80: Flags [P.], seq 1:118, ack 1, win 1460, options [...], length 117
5     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], ack 118, win 181, options [...], length 0
6     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq 1:2897, ack 118, win 181, options [...], length 2896
```

```
7     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
8     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
9     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
10     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
11     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
12     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
```

Вывод TCPDUMP#2 (на eth0 deb-serv-03):

Код: Выделить всё

```
1     IP 172.16.5.3.57925 > 192.168.0.1.80: Flags [S], seq
1723325723, win 5840, options [mss 1460...], length 0
2     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [S.], seq
2482933888, ack 1723325724, win 5792, options [mss 1460...],
length 0
3     IP 172.16.5.3.57925 > 192.168.0.1.80: Flags [.], ack 1,
win 1460, options [...], length 0
4     IP 172.16.5.3.57925 > 192.168.0.1.80: Flags [P.], seq
1:118, ack 1, win 1460, options [...], length 117
5    IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], ack 118,
win 181, options [...], length 0
6     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
7      IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
8     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1449:2897, ack 118, win 181, options [...], length 1448
9      IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
10     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
11      IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
12     IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
```

```
13     IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
14    IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
15     IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
16    IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
17     IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
18    IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
19     IP 172.16.250.2 > 192.168.0.1: ICMP 172.16.5.3
unreachable - need to frag (mtu 1400), length 556
20    IP 192.168.0.1.80 > 172.16.5.3.57925: Flags [.], seq
1:1449, ack 118, win 181, options [...], length 1448
```

Как видите, ситуация вполне ожидаемая. Первые 6 строк в каждом выводе точно такие же, как и при нормальной передаче (см. Описание в предыдущем примере). Но вот дальше начинаются расхождения. ICMP 3:4 точно так же генерируется на deb-serv-03(строки 7, 9 11.13, 15, 17, 19 в TCPDUMP#2), но deb-serv его не получает и продолжает слать пакеты размером в 1500 байт(строки с 6 по 12 в TCPDUMP#1 и 6, 8, 10, 12, 14, 16, 18 и 20 в TCPDUMP#2). С каждым разом время между повторной посылкой все увеличивается (в данных примерах я отбросил временые метки, но на самом деле это так). Никаких данных размером большим, чем PMTU, в таком случае не передать. Но увы, TCP этого не знает и продолжает слать пакеты с MSS, выбранным в момент установки соединения. Именно эта ситуация и называется Path MTU Discovery Black Hole (Черная дыра в определении транспортного MTU). Я постарался представить ее в упрощенном виде на рис. 3.

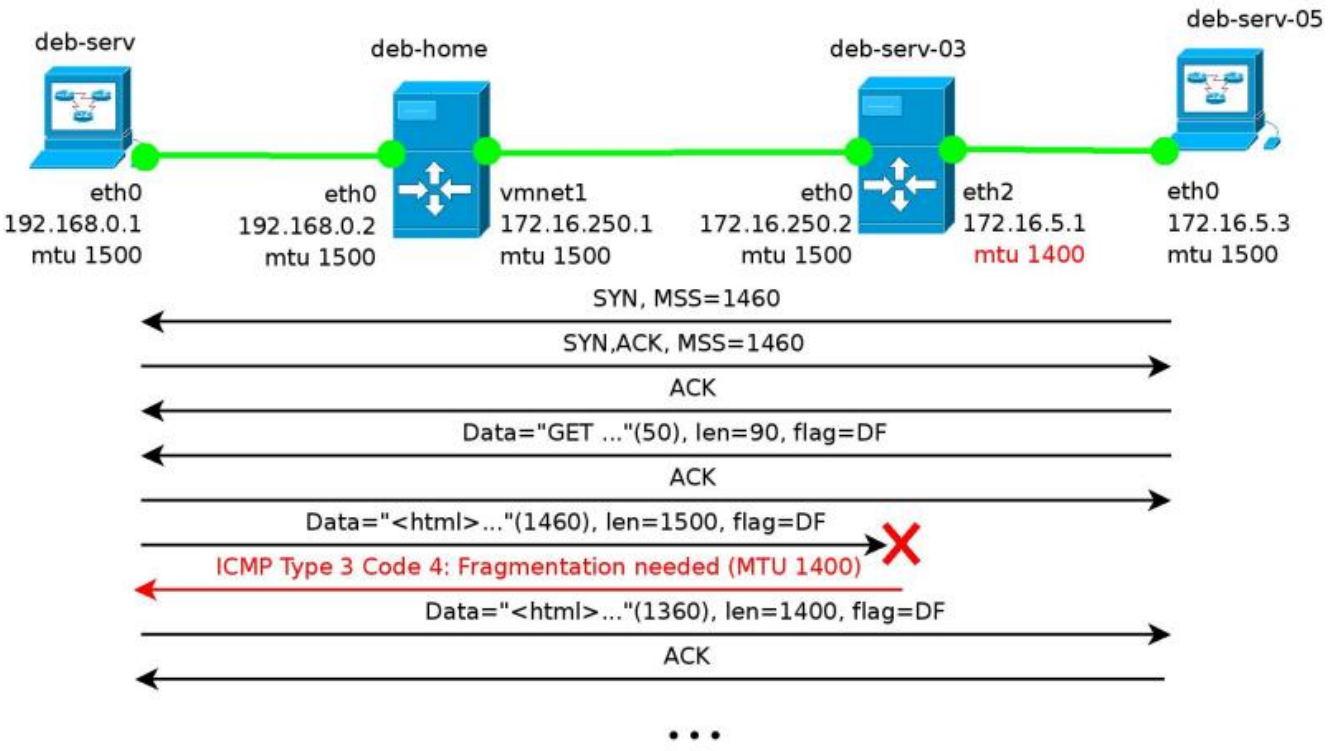

Рис. 3. Черная дыра в определении PMTU.

Эта проблема совсем не нова. Она описана в RFC 2923 (http://tools.ietf.org/html/rfc2923) в 2000 году. Но тем не менее, продолжает встречаться с завидным упорством у многих провайдеров. А ведь именно провайдер виноват в данной ситуации: не нужно блокировать ICMP тип 3 код 4. Причем слушаться «голоса разума» (т. е. клиентов, понимающих в чем проблема) они обычно не хотят. Поэтому не будем обращать на них внимание и попробуем решить проблему, исходя из собственных средств.
Разработчики Linux, тоже знающие об этой проблеме, предусмотрели специальную опцию в iptables. Цитата из man iptables:

> *TCPMSS*
> *This target allows to alter the MSS value of TCP SYN packets, to control the maximum size for that connection (usually limiting it to your outgoing interface's MTU minus 40 for IPv4 or 60 for IPv6, respectively). Of course, it can only be used in conjunction with -p tcp. It is only valid in the*

mangle table. This target is used to overcome criminally braindead ISPs or servers which block “ICMP Fragmentation Needed” or “ICMPv6 Packet Too Big” packets. The symptoms of this problem are that everything works fine from your Linux firewall/router, but machines behind it can never exchange large packets:
1) Web browsers connect, then hang with no data received.
2) Small mail works fine, but large emails hang.
3) ssh works fine, but scp hangs after initial handshaking.
Workaround: activate this option and add a rule to your firewall configuration like:
*iptables -t mangle -A FORWARD -p tcp –tcp-flags SYN,RST SYN *
-j TCPMSS –clamp-mss-to-pmtu–set-mss value
Explicitly set MSS option to specified value.
–clamp-mss-to-pmtu
Automatically clamp MSS value to (path_MTU – 40 for IPv4; -60 for IPv6).

These options are mutually exclusive.

Мой вольный перевод:

TCPMSS
Это действие позволяет изменять значение MSS в TCP SYN пакетах, для контроля максимального размера пакетов в этом соединении (Обычно ограничивая его MTU исходящего интерфейса минус 40 байт для IPv4 или минус 60 для IPv6). Конечно, это действие может использоваться только в сочетании с -p tcp. Разрешено это только в таблице mangle. Это действие используется для преодоления преступной некомпетентности провайдеров и серверов, блокирующих “ICMP Fragmentation Needed” или “ICMPv6 Packet Too Big” пакеты. Симптомы этой проблемы – все прекрасно работает на вашем сетевом экране или роутере, но машины за ним никогда не смогут обмениваться большими пакетами:
1) Веб браузеры связываются, но просто висят без пересылки данных.
2) маленькие электронные письма приходят нормально, но большие висят.
3) ssh работает отлично, но scp висит после начальных рукопожатий(прим пер: процесс установки TCP соединения также

называют “тройным рукопожатием”).
Решение: активировать эту опцию и добавить правило, подобное нижеприведенному, в конфигурацию своего сетевого экрана:
*iptables -t mangle -A FORWARD -p tcp –tcp-flags SYN,RST SYN *
-j TCPMSS –clamp-mss-to-pmtu–set-mss значение
Явная установка в опции MSS специфического значения.
–clamp-mss-to-pmtu
Автоматическая установка значения MSS в (path_MTU – 40 для IPv4; -60 для IPv6).

Эти опции являются взаимоисключающими.

Как видите, много всего написали, даже описали примерные симптомы проблемы. А такое поведение провайдеров назвали “преступной некомпетентностью(criminally braindead)”, в чем я с ними полностью согласен. Давайте исследуем, как же будет работать эта опция в нашем примере. Добавляем на deb-serv-03 рекомендованное правило:

Код: Выделить всё

```
iptables -t mangle -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 1360
```

И смотрим что получилось:
Вывод TCPDUMP#1 (на eth0 deb-serv):

Код: Выделить всё

```
1    ip 172.16.5.3.33792 > 192.168.0.1.80: flags [s], seq 1484543117, win 5840, options [mss 1360...], length 0
2    ip 192.168.0.1.80 > 172.16.5.3.33792: flags [s.], seq 2230206317, ack 1484543118, win 5792, options [mss 1460...], length 0
3    ip 172.16.5.3.33792 > 192.168.0.1.80: flags [.], ack 1, win 1460, options [...], length 0
4    ip 172.16.5.3.33792 > 192.168.0.1.80: flags [p.], seq 1:118, ack 1, win 1460, options [...], length 117
5    ip 192.168.0.1.80 > 172.16.5.3.33792: flags [.], ack 118, win 181, options [...], length 0
6    ip 192.168.0.1.80 > 172.16.5.3.33792: flags [.], seq
```

```
1:2697, ack 118, win 181, options [...], length 2696
7     ip 172.16.5.3.33792 > 192.168.0.1.80: flags [.], ack
1349, win 2184, options [...], length 0
8     ip 192.168.0.1.80 > 172.16.5.3.33792: flags [.], seq
2697:5393, ack 118, win 181, options [...], length 2696
9     ip 192.168.0.1.80 > 172.16.5.3.33792: flags [fp.], seq
5393:6380, ack 118, win 181, options [...], length 987
10     ip 172.16.5.3.33792 > 192.168.0.1.80: flags [.], ack
2697, win 2908, options [...], length 0
```

Вывод TCPDUMP#3 (на eth0 deb-serv-05):

Код: Выделить всё

```
1     IP 172.16.5.3.33792 > 192.168.0.1.80: Flags [S], seq
1484543117, win 5840, options [mss 1460...], length 0
2     IP 192.168.0.1.80 > 172.16.5.3.33792: Flags [S.], seq
2230206317, ack 1484543118, win 5792, options [mss 1360...],
length 0
3     IP 172.16.5.3.33792 > 192.168.0.1.80: Flags [.], ack 1,
win 1460, options [...], length 0
4     IP 172.16.5.3.33792 > 192.168.0.1.80: Flags [P.], seq
1:118, ack 1, win 1460, options [...], length 117
5     IP 192.168.0.1.80 > 172.16.5.3.33792: Flags [.], ack 118,
win 181, options [...], length 0
6     IP 192.168.0.1.80 > 172.16.5.3.33792: Flags [.], seq
1:1349, ack 118, win 181, options [...], length 1348
7     IP 192.168.0.1.80 > 172.16.5.3.33792: Flags [.], seq
1349:2697, ack 118, win 181, options [...], length 1348
8     IP 172.16.5.3.33792 > 192.168.0.1.80: Flags [.], ack
1349, win 2184, options [...], length 0
9     IP 172.16.5.3.33792 > 192.168.0.1.80: Flags [.], ack
2697, win 2908, options [...], length 0
10     IP 192.168.0.1.80 > 172.16.5.3.33792: Flags [.], seq
2697:4045, ack 118, win 181, options [...], length 1348
```

Разбираем:

1. В строках 1-3 мы уже привычно наблюдаем установку TCP соединения. Но обратите внимание на значения MSS. В TCPDUMP#1 от deb-serv-05 приходит значение 1360, в то время как в TCDUMP#3 видно, что уходит пакет с MSS=1460. Именно так и работает правило с –set-mss 1360. Оно редактирует значение MSS

у пролетающих пакетов. Для SYN пакета, пришедшего в ответ, это значение тоже отредактировано.
2. В строках 4 и 5 обоих выводов мы опять наблюдаем отправку GET запроса и подтверждение получения.
3. В строке 6 для TCPDUMP#1 и строках 6 и7 для TCPDUMP#3 видим отправку пакетов с данными, но теперь размер каждого из пакетов не превышает 1400 байт. Опять происходит странный глюк с TCPDUMP#1, где виден один большой пакет, в то время как в TCPDUMP#3 мы наблюдаем приход 2-х пакетов.
4. Дальнейший обмен пакетами идет в соответствии с правилами протокола TCP. Но ни разу размер пакета на превышал 1400 байт.

В упрощенном виде поведение MSS представлено на рис. 4. Я не стал показывать обмен данными, так как он аналогичен обычному поведению.

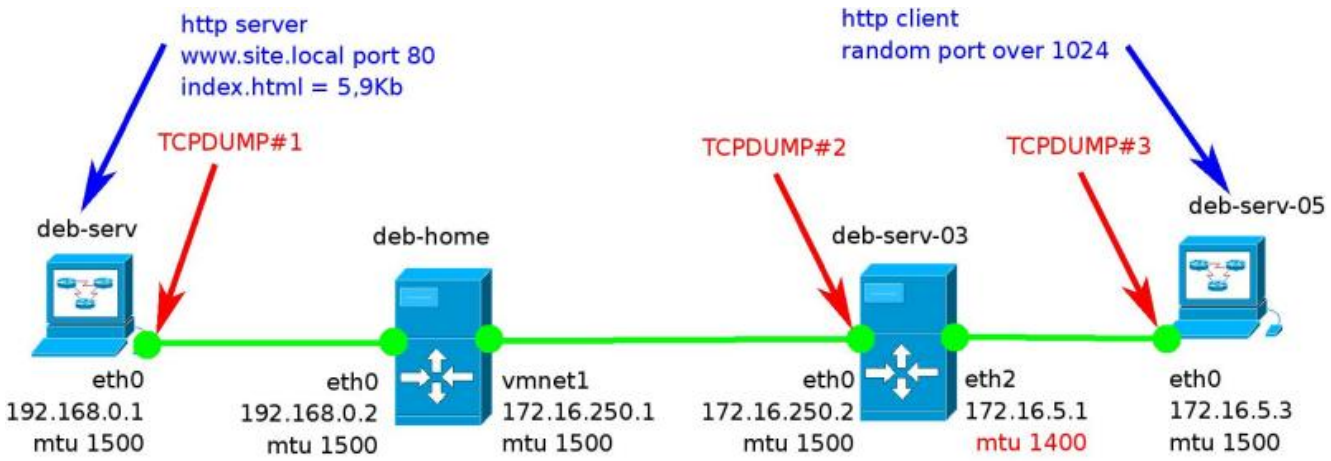

Рис. 4. Изменение MSS на лету.

Хотя в man iptables описаны две опции, но я пока применил только одну. Нужная опция зависит от конкретной ситуации. Все ситуации можно разделить на 2 типа:

1. На вашем маршрутизаторе сайты открываются нормально, у клиентов в локальной сети наблюдаются проблемы.
В этом случае наименьший MTU на всем пути находится именно на вашем сервере. Обычно это некие протоколы инкапсуляции, типа PPPoE, PPtP и тд. Для данной ситуации лучше всего подойдет

опция —-clamp-mss-to-pmtu, которая автоматически установит минимальный MSS на все транзитные пакеты.

2. На вашем маршрутизаторе и у клиентов в локальной сети сайты не открываются.
В таком случае наименьший MTU находится где-то у провайдера и вычислить его стандартными средствами сложновато. Специально для этого я написал скрипт на python, который поможет определить необходимый размер MSS для данной ситуации:

```
#!/usr/bin/env python
# -*-coding: utf-8 -*-
# vim: sw=4 ts=4 expandtab ai

import socket
import os
import time
import sys

# Полное имя веб сервера на котором проводятся испытания. Следует выбирать из
# сайтов, которые точно не работают.
HOST = 'www.site.local'
# Временной интервал, в течении которого следует ожидать ответа от сайта.
# Слишком маленькое значение может породить ложные срабатывания, слишком
# большое - долгое время работы скрипта.
TIMEOUT = 25.0
# Количество байт, которые надо получить с веб сервера, чтобы убедится что он
# наверняка работает. Рекомендуется устанавливать большим нежели значение MTU
BUF = 3000
# Значение MTU на интерфейсе в интернет.
MTU = 1500
# Значение MSS будет искаться в пределе от MTU-LIM-40 до MTU-40. Запрещено
# ставить значение больше MTU и не рекомендуется ставить значения более чем
```

```
# 100-200 - это может привести к большому времени работы
скрипта.
LIM = 100
# Задержка между обращениями к сайту. Рекомендуется
устанавливать отличной от
# нуля на медленном канале.
TRY_TIME = 0

def set_mss(mss, action='A'):
    return os.system("iptables -t mangle -%s OUTPUT -p tcp --
tcp-flags \
             SYN,RST SYN -j TCPMSS --set-mss %d" % (action,
mss) )

def check_connection(host):
    s = socket.socket()
    s.connect( (host, 80) )
    s.send('GET / HTTP/1.1\r\nHost: %s\r\n\r\n' % host)
    s.settimeout(TIMEOUT)
    try:
        d = len( s.recv(BUF) )
    except:
        d = 0
    s.close()
    return d

def main():
    mss = MTU - 40
    if not check_connection(HOST):
        mss = MTU - 40 - LIM
        set_mss(mss)
        if not check_connection(HOST):
            set_mss(mss,'D')
            print "Error: Too small LIM"
            sys.exit(1)
        else:
            while check_connection(HOST):
                time.sleep(TRY_TIME)
                set_mss(mss,'D')
                if mss >= MTU-40:
                    print "Error in determining MSS"
```

```
                    sys.exit(1)
                mss += 1
                set_mss(mss)
            set_mss(mss,'D')
            mss -= 1
    print 'MSS = %d' % (mss)

if __name__ == '__main__':
    main()
    sys.exit(0)
```

Запускать скрипт нужно с правами суперпользователя. Алгоритм его работы таков:
1. Пытаемся получить некоторое количество данных с сайта с нормальным значением MSS.
2. Если это не получается, то понижаем MSS на iptables цепочке OUTPUT до MTU – 40 – LIM.
3. Если и после этого мы не можем получить данные, то выдаем ошибку о том, что LIM имеет слишком маленькое значение.
4. Последовательно наращивая MSS, ищем тот момент, когда данные перестанут поступать. После этого выводим последнее рабочее значение MSS.
5. Если мы дошли до MSS=MTU-40, то выводим ошибку о том, что не можем определить MSS. Данная ситуация является ошибочной, т. к. в пункте 1 проводим аналогичную проверку, и, если результаты не совпадают, это повод задуматься.

После получения нужного MSS необходимо вписать его в соответствующее правило (используя –set-mss). Можно обойтись и без скрипта, на глаз понизив значение MSS, но лучше выяснить его точно – меньше накладые расходы на пересылку пакетов.

Внимание! Чтобы использовать эти правила для iptables в ядре должны быть включены две опции – CONFIG_NETFILTER_XT_TARGET_TCPMSS и CONFIG_NETFILTER_XT_MATCH_TCPMSS. Проверить можно например так:

```
cat /boot/config-2.6.26-2-686 | grep TCPMSS
```

```
CONFIG_NETFILTER_XT_TARGET_TCPMSS=m
CONFIG_NETFILTER_XT_MATCH_TCPMSS=m
```

Как видите у меня они подключены модулями.

Часто на форумах можно встретить советы понизить MTU на том или ином интерфейсе. Нужно понимать, что это не панацея, и результат зависит от того, на каком интерфейсе понижать. Если понизим на одном из интерфейсов участников TCP соединения, то это принесет эффект, так как заявленная MSS будет соответствовать минимальному размеру пакета. Но если это будут не конечные точки, а один из транзитных маршрутизаторов, то без включения опции –clamp-mss-to-pmtu никакого эффекта не будет.

Надеюсь данная статья поможет Вам решить подобную проблему как у себя, так и у ваших друзей и знакомых. **Еще раз обращаюсь к специалистам провайдеров – НЕ БЛОКИРУЙТЕ ICMP ТИП 3 КОД 4 – этим вы создаете проблемы вашим колегам.**

https://unixforum.org/download/file.php?id=21359&mode=view