

Москва

ноябрь 2005

Алексей Барабанов <alekseybb at mail dot ru>.

TCP поверх TCP не такая уж плохая идея!

Есть расхожее мнение, что сетевые туннели выгоднее делать на основе протоколов низкого уровня с минимальными размерами заголовков и очень простым протоколом. Считается, что TCP, как несущий протокол, создает много проблем. Так ли это?

Введение.

Использование TCP стало столь привычным, что большинство просто не задумывается о заложенных в этот протокол механизмах и во всех случаях полагается на «интеллект» системы. Одновременно с этим, если встречаются какие-то сведения, объясняющие что-то в его поведении, или теории, построенные на основе, так сказать, «упрощенных» представлений, то у многих просто не достаёт знаний правильно оценить полученную информацию. Что приводит к появлению технологических мифов. Один из которых, порожденный статьей [1] Олафа Титца (Olaf Titz), здесь и попробуем опровергнуть. К сожалению, в России все, что написано на иностранном языке, да еще размещено на зарубежном сайте, воспринимается, как святое откровение. Хотя, многие далее вынесенного в ее заголовок тезиса «Why TCP Over TCP Is A Bad Idea» и не читают. Чтобы уравнивать шансы тех, кто имеет затруднения с английским языком, предложим дословный технический перевод упомянутого опуса [2]. В переводе не использовались никакие литературные экстраполяции, чтобы максимально точно донести идеи автора и ни в коем случае не исправить на этапе перевода что-то из многочисленных его ошибок. Разберем текст этой статьи.

Суть проблемы.

Итак, первое, что по мнению Олафа мешает эффективной работе TCP в качестве туннеля, это повторные передачи. Если отбросить всякие туманные рассуждения о насыщении канала (meltdown) из-за мифического фиксированного таймаута, которой нет в TCP, то в «сухом остатке» в первом разделе статьи лишь утверждение об экспоненциальном росте тайм-аута, если, цитирую, «сегмент задерживается сверх тайм-аута». Вот тут рассмотрим подробнее. Автор упорно не желает использовать общепринятую терминологию и предпочитает изъясняться как колдун-друид. Но так как в тексте упоминается RFC2001, то можно воспользоваться ссылкой [3] и попытаться догадаться, на что намекает Олаф Титц. Единственный показатель, имеющий в RFC2001 экспоненциальный рост, это CWND (congestion window). Но окно насыщения, или CWND, связано с пропускной способностью прямой зависимостью. Автор же намекает, что задержки растут по экспоненте и это якобы приводит к снижению темпа передачи. Задержки это RTT, или время обращения (round trip time), на основании которых высчитывается RTO, или таймер повтора (retransmission time out). Об это нет ничего в RFC2001. Но по другим документам можно узнать, как это происходит. Например, в RFC793 [4], упоминаемом также в статье, описан метод расчета RTO для каждого текущего пакета. Так вот там используется показатель SRTT, или взвешенное время обращения (smoothed round trip time). Подчеркиваю – взвешенное! Где тут Олаф «раскопал» экспоненту? Конечно, можно догадаться, что автор спутал расчет RTO при задержке с расчетом RTO при потере! Вот, если сегмент вообще пропал, и от

получателя нет никаких ответов, тогда алгоритм TCP на самом деле требует произвести повторы с экспоненциальным замедлением. Но, не смущаясь мелкими деталями, Олаф связывает увеличение периодов проверки на обрыв соединения с борьбой с насыщением, то есть с тем самым meltdown-ом, который как ружье со стены в известной пьесе, если упомянуто в первом акте, то далее обязательно «выстрелит». Таким образом, поскольку экспоненциального роста задержек передачи в TCP нет, то измышления первой части обсуждаемой статьи [1] не имеют отношения к реализации TCP в нашей с Вами Вселенной.

Рассуждения Олафа, где он считает, что трафик TCP в Интернете регулируется тайм-аутами, просто выкинем, как не соответствующие реальности и потому не актуальные. Конечно, можно было бы везде, где в тексте встречается «timeout», при переводе подставить «таймер», что сильно приблизит многие рассуждения к действительному положению дел, но тогда пришлось бы исправить и многое другое. Но, как уже было сказано выше, перевод сделан самым безжалостным способом.

Второе, что смутило Олафа Титца, это взаимодействие транспортного TCP и транспортируемого. Для ясности обратимся к схеме инкапсуляции, которая обсуждается в [1]. Скопируем ее в переводном варианте и добавим немного комментариев.

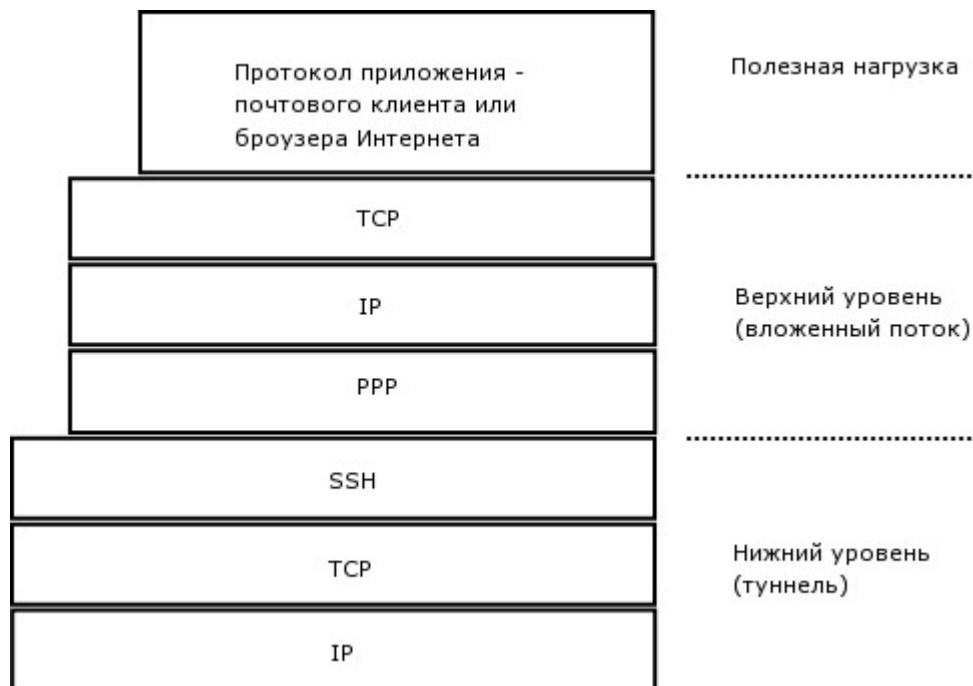


Рисунок 1. Схема инкапсуляции.

Автор утверждает, что эти два потока TCP (см. рисунок 1) могут иметь разные таймеры. Дословно: «может так случиться, что соединение нижнего уровня имеет медленные таймеры, возможно как остаток с периода медленного или ненадежного основного соединения». То есть автор предположил, что вложенное TCP успело разогнаться, а несущее, тем не менее, оставалось медленным. Это просто невозможно, так как противоречит законам сохранения, действующим в нашей Вселенной. Надо признать, что именно здесь Олафа «понесло». Он увлеченно живописует, как внутренний слой, такой быстрый и такой «несчастный», будет, не получая подтверждений, посылать пакет за пакетом и тем самым – вот оно, старое ружье выстрелило – создаст «внутренний meltdown эффект». Здесь замечу, что термин «внутренний meltdown эффект» изобретен просто «на

ходу». Жаль разочаровывать Олафа, но после неподтверждения пакета и в отсутствии вообще всяких подтверждений TCP начинает процедуру медленного старта, в ходе которой поток TCP снова будет искать предельную передающую возможность среды, постепенно увеличивая скорость передачи, начиная с самых минимальных значений. Иначе говоря, TCP не снижает темп передачи, а вообще прекращает ее и затем возобновляет снова со стартовых значений. И это, кстати, дает гарантию, что внутренний TCP никогда не будет работать быстрее внешнего того, что образует туннель. Так как, если параметры внутреннего TCP приведут к тому, что он разгонится быстрее туннельного, то после же столкновения он сбросится снова к состоянию медленного старта и снова начнет «догонять» туннельный TCP.

Давайте представим, что реально может случиться из-за потерь сегментов (именно так и следует далее их называть, а не пакетами) в таком туннеле. Ну, во-первых, просто обрыв связи будет обработан и тем и другим TCP соответственно и в пределах перечисленных RFC. А вот с повторами, и правда, могут возникнуть некоторые проблемы. Покажем на схеме (рисунок 2), что будет происходить, если в пути потеряется сегмент с данными.

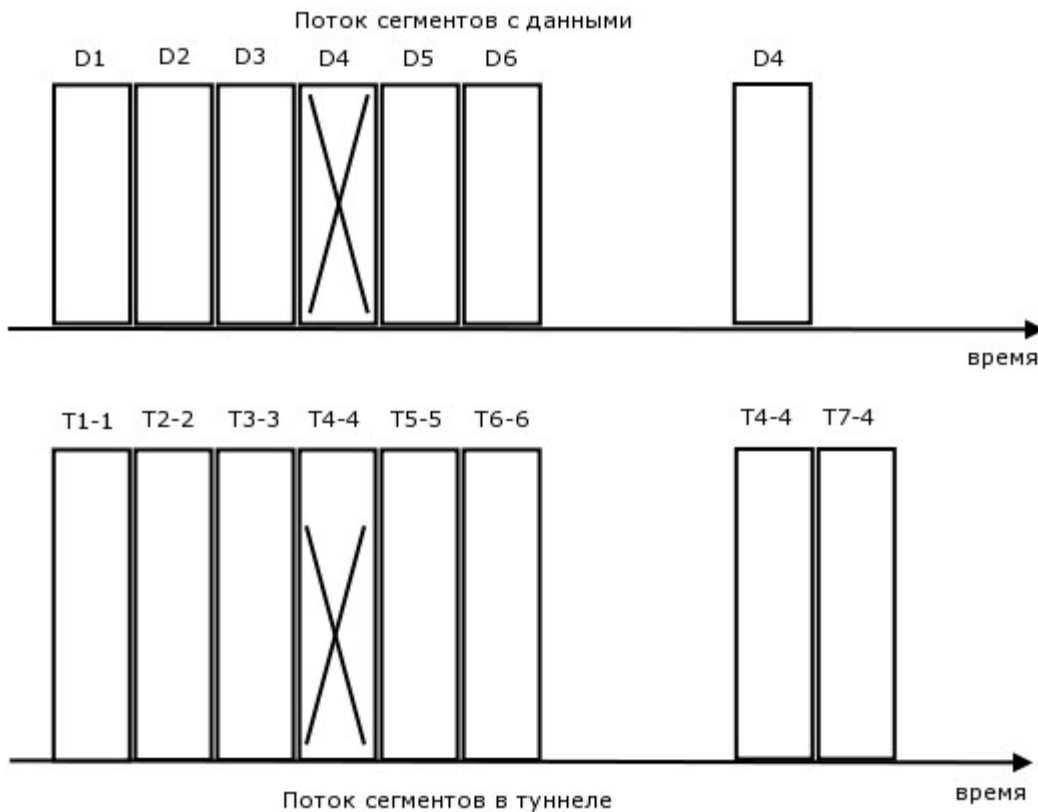


Рисунок 2. Последствия потери сегмента.

Предположим, что пропал сегмент D4. Тогда оба стека протоколов получают повторный ACK с последним принятым номером и согласно алгоритму Fast Retransmission, или быстрой повторной передачи, пропущенный пакет будет повторен. На рисунке изображена ситуация, когда оба стека имеют одинаковый RTT и, значит, близкие RTO. Тогда транспортный TCP сформирует повтор пропавшего сегмента T4-4 и точно также добавит в поток сегмент T7-4, содержащий сегмент с данными D4, повторенный в транспортируемом TCP. Это самый худший вариант развития событий. Иначе говоря, если потери канала составляют 10%, то из полезной емкости канала будет изъято уже 20%. Но, как уже замечено, это предельно плохой вариант. Поскольку RTT туннеля и RTT

транспортируемого TCP на практике не совпадают, то, скорее всего, реальные потери полосы скорее всего гораздо меньше.

Может показаться, что в этих рассуждениях упущена ситуация, когда будет потерян не сегмент с данными, а его подтверждение. Но если это произойдет, все случится точно также, как и на рисунке 2. Для TCP нет разницы между потерянным сегментом и потерянным подтверждением, если и те и другие высылаются не так, как рассуждает Олаф, а сериями в пределах размера CWND. Но с другой стороны, теперь уже для TCP трафика происходит теоретическое учетверение потерь, то есть из полезной емкости канала будет изыматься уже 40%. Хотя, это лишь «на бумаге», то есть снова здесь приведен самый плохой вариант.

Теперь, глядя на рисунок 2, ответьте, в каком TCP надо запретить повторы? Олаф Титц делает вывод, что повторы надо запретить в верхнем транспортируемом TCP. На самом деле, повторы следует запретить в нижнем, туннельном TCP. Во-первых, потому что туннель это одно TCP соединение, а вложенных может быть множество. Во-вторых, потому что туннель создается на специальном хосте, а внутренние потоки соединяют разные клиентские компьютеры со стандартными TCP стеком. И, наконец, в-третьих, кроме TCP есть еще много разных протоколов, сегменты с которыми все-таки очень не бесполезно восстановить при потере, а отличить их от TCP может только стек туннеля. Но такое расширение TCP туннелей пока не реализовано. Здесь лишь рассматривается гипотетическая возможность. Но ничего не мешает в практической ситуации для пробы запретить повторы через параметры `sysctl` линуксового ядра на тех хостах, между которыми поднят туннель, и проверить действенность такой настройки.

Итак, и вторая часть рассуждений о проблемах TCP не выдерживает элементарной проверки. Быть может, в разделе «практический опыт» нам откроется истина? Рассмотрим и его по-внимательнее.

В практической части статьи [1] содержится информация, которая может стать разгадкой всех проблем Олафа Титца. Оказывается, у автора этой статьи, цитирую, «использовалась волоконно-оптическая связь, которая страдала частыми потерями пакетов, иногда 10-20%». Странная связь! Вероятно, на карманных фонариках. Последнее конечно шутка, но, тем не менее, это позволяет точно определить условия применения тезисов автора упомянутой статьи. Обратимся к RFC 2001 [3]. Там однозначно установлено, что все алгоритмы TCP рассчитаны при допущении, что потери в канале составляют менее 1%. Вероятно, у Олафа были проблемы с кабельщиком. Быть может, суровому монтеру не понравилась его прическа [5] и он решил таким путем выразить свое возмущение. И Олаф утверждает, что, создав CIPE [6], смог решить проблемы и обеспечить надежную работу на канале с потерями до 20%. И если не проверить утверждения автора с помощью тестов с туннелем на подобном канале, то получится, что здесь предлагается рассуждения Олафа Титца заменить рассуждениями Алексея Барабанова. Лишь опыт позволит рассудить, на чьей стороне правда (или правдоподобность). Поскольку нет простой возможности создать искусственную линию связи с характеристиками похожими на те, что создали так много проблем Олафу, то заменим реальные, обычно модулированные, помехи случайными. И дополнительно ограничим их 10%-ми. Думаю, что эта замена не помешает понять суть происходящих процессов, так как на практике и 10% и 20% значительно больше указанного в RFC 1%, и отличаются лишь временем, затраченным на проведение эксперимента, поскольку чем больше потери, тем меньше скорость передачи.

Тестовая сеть.

Проверку будем проводить путем передачи тестового массива псевдослучайных данных в одном направлении через TCP и UDP туннели. Это конечно примитивная модель взаимодействия, но только так можно в чистом виде попытаться определить зависимость характеристик канала и свойств полученного потока данных. Дополнительно в центре маршрута ограничим полосу, например до 10 Мбит, и внесем случайные потери в трафик. Конечно, 10 Мбит значительно выше типичной скорости канала, предоставленного ISP для подключения к Интернету. Но иначе придется уменьшить объем пересылаемых данных, чтобы в приемлемое время провести эксперимент. Перечислим элементы, которые нужны для организации таких проверок. Во-первых, хост-отправитель и хост-получатель трафика. Во-вторых, два хоста, между которыми проложен туннель. Поскольку «удаленный» конец туннеля может совпадать с хостом-получателем, то кроме перечисленных трех хостов нужен еще один, где будет эмулироваться «узкое место» в сети. Итого, достаточно будет 4 компьютера соединить последовательно. Строго говоря, хост-отправитель можно совместить с «ближним» концом туннеля. Но надо учесть, что вход в туннель тоже является своего рода ограничителем трафика, или шейпером, поэтому пусть будут в тестовой сети четыре компьютера, которые назовем wstovert, wsalekseybb, wskostja и server. Общий вид полученной сети представлен на рисунке 3, и дальнейший комментарий будем вести согласно изображенной там схемы.

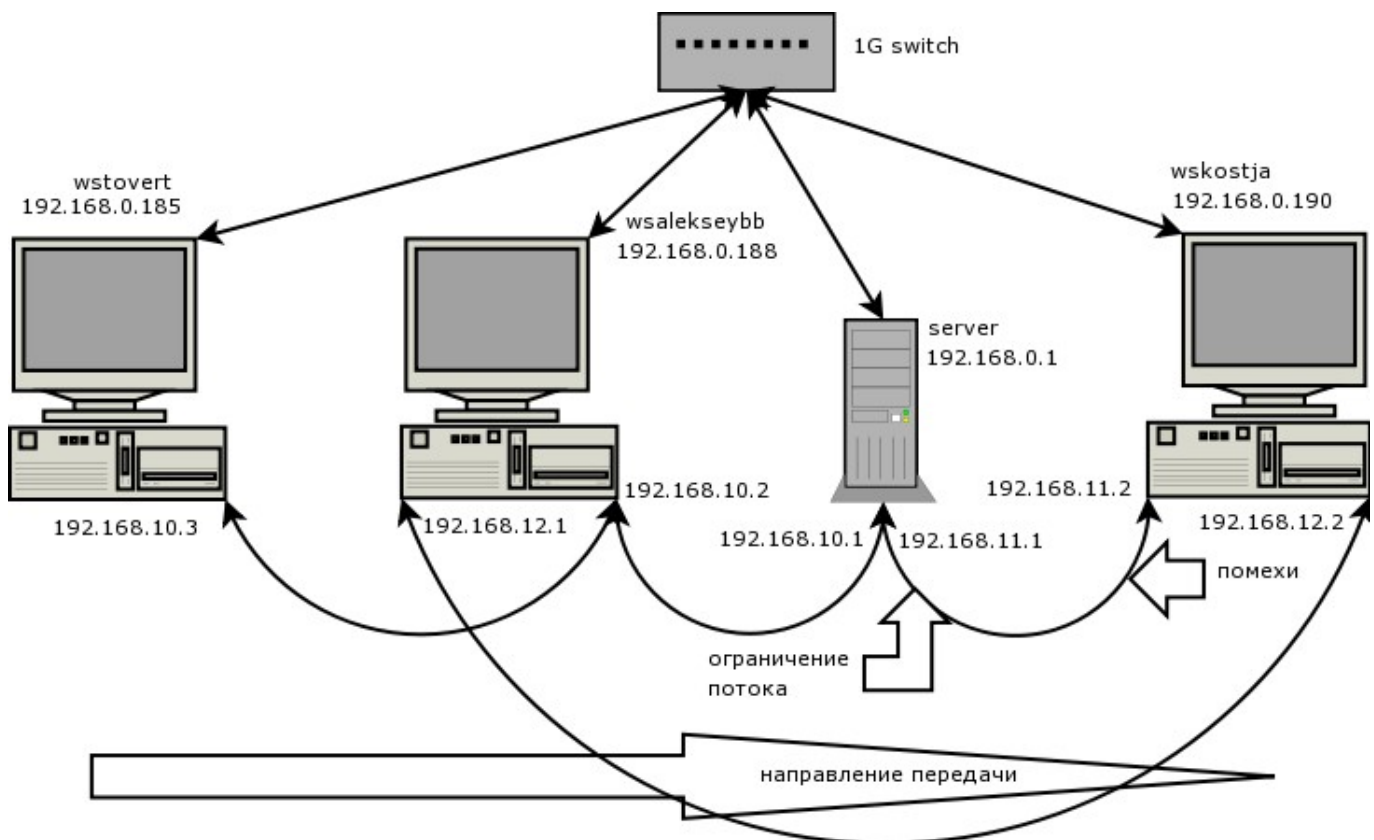


Рисунок 3. Схема тестовой сети.

Все компьютеры включены в общую сеть 192.168.0.0/24, объединены общей кабельной системой с помощью коммутатора 1 Гбит и обмениваются данными по протоколу 1000BaseTX, то есть значительно выше проверяемых скоростей передачи, что позволит максимально снизить систематическую ошибку. На эту сеть наложены две виртуальные

сети. Первая из них, с адресом 192.168.10.0/24, включает wstover, wsalekseybb и server, а вторая, с адресом 192.168.11.0/24, только server и wskostja. Практически все компьютеры могут определять адреса друг друга с помощью arp и обмениваться пакетами через общий коммутатор. Но нам ведь надо обеспечить передачу трафика между компьютерами так, будто они включены последовательно один за другим. Поэтому с помощью правил маршрутизации и преобразования адресов заставим трафик от wstover проходить до wskostja через wsalekseybb и server так, как указывает стрелка на рисунке 3. Для этого на каждом компьютере укажем статический маршрут до следующей по ходу движения стрелки на рисунке 3 сети через соседний компьютер, а на том включим NAT так, чтобы заменять в таком маршруте адрес отправителя на собственный. Окончательно должно получиться так:

```
wstover:~ # traceroute 192.168.11.2
traceroute to 192.168.11.2 (192.168.11.2), 30 hops max, 40 byte packets
 1  192.168.10.2  0.000 ms   0.000 ms   0.000 ms
 2  192.168.10.1  0.566 ms   0.086 ms   0.088 ms
 3  192.168.11.2  0.145 ms   0.170 ms   0.241 ms
wstover:~ #
```

Поскольку главный вопрос заключается в изучении поведения инкапсулированного трафика, то в тестовой сети настроим также и туннель. Воспользуемся программным обеспечением, которое позволит поднять туннель, как на основе UDP транспорта, так и TCP. По этой причине CIPR [6] не подходит. Выберем OpenVPN [7]. Тем более, что согласно сравнительной таблице [8] этот проект рекордсмен в разделе Popularity. Туннельная сеть образует виртуальное соединение между wsalekseybb и wskostja, определенное как сеть 192.168.12.0/24. Для станции wsalekseybb настройки OpenVPN в режиме клиента TCP-туннеля будут производиться согласно следующему файлу конфигурации:

```
wsalekseybb:~ # cat /etc/openvpn/test.conf
proto tcp-client
tcp-queue-limit 1000
remote 192.168.11.2
dev tun
port 5000
tcqueuelen 1000
secret /etc/openvpn/static.key
ifconfig 192.168.12.1 192.168.12.2
route 192.168.12.0 255.255.255.0 192.168.12.1
wsalekseybb:~ #
```

Для станции wskostja все аналогично, меняются лишь адреса, протокол на tcp-server и удаляется строка remote, что переводит туннель в состояние прослушивания. В режиме UDP на обоих концах протокол просто переключается в udp и удаляется параметр tcp-queue-limit. И теперь путь между wstover до wskostja проходит через туннель.

```
wstover:~ # traceroute 192.168.12.2
traceroute to 192.168.12.2 (192.168.12.2), 30 hops max, 40 byte packets
 1  192.168.10.2  0.000 ms   0.000 ms   0.000 ms
 2  192.168.12.2  7.088 ms   5.756 ms   0.787 ms
wstover:~ #
```

Чтобы доказать, что сам туннель проходит через server, проверим путь до удаленного конца туннеля от wsalekseybb:

```
wsalekseybb:~ # traceroute 192.168.11.2
traceroute to 192.168.11.2 (192.168.11.2), 30 hops max, 40 byte packets
 1  192.168.10.1  0.195 ms    0.106 ms    0.120 ms
 2  192.168.11.2  0.233 ms    0.000 ms    0.000 ms
wsalekseybb:~ #
```

Таким образом, можно, посылая трафик со станции wstoverт до станции wskostja, осуществлять на промежуточном хосте server нужные манипуляции в соответствии с исследуемой моделью канала.

Для работы создадим на всех участвующих в экспериментах хостах специального пользователя tovert и разместим в его домашних директориях в файле .ssh/authorized_keys2 специально подготовленные ключи, чтобы можно было выполнять необходимые действия в автоматическом режиме с одной из станций. И для удобства разрешим через sudo выполнение пользователем tovert команд с привилегиями суперпользователя без ввода пароля.

На wstoverт подготовим файл с данными, которые будут посылаться в тестовых сеансах:

```
# ssh tovert@wstoverт "dd if=/dev/urandom of=~/.100M.bin bs=1024 count=102400"
```

Симуляцию TCP трафика будем создавать с помощью команды, отправляющей 100Мбайт из файла на целевой хост. В качестве слушателя TCP используем sshd, который перешлет все принятые данные в /dev/null, чтобы не помешала буферизация. Причем, для сокращения затрат на загрузку файла с диска, каждую тестовую отправку будем предварять локальным копированием в /dev/null:

```
# dd if=/home/tovert/100M.bin bs=1024 count=102400 >>/dev/null
# dd if=/home/tovert/100M.bin bs=1024 count=102400 | ssh tovert@192.168.12.2 "cat - >>/dev/null"
```

Симуляция UDP трафика будет производиться утилитой netcat, точно также предварительно произведя локальное копирование:

```
# dd if=/home/tovert/100M.bin bs=1024 count=102400 >>/dev/null
# dd if=/home/tovert/100M.bin bs=1024 count=102400 | netcat -vu -w 1 192.168.12.2 22222
```

На приемной стороне UDP трафик будет приниматься специальным слушателем и подсчитываться:

```
# netcat -vlnu -s 192.168.12.2 -w 15 -p 22222 | wc -c
```

В точке прохождения трафика через хост server включим ограничитель полосы типа НТВ (Hierarchical Token Buckets) на 10 Мбит со следующей конфигурацией:

```
qdisc htb 1: r2q 10 default 0 direct_packets_stat 0
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
class htb 1:1 root rate 10Mbit ceil 10Mbit burst 14704b cburst 14704b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
 lended: 0 borrowed: 0 giants: 0
 tokens: 9191 ctokens: 9191

class htb 1:10 parent 1:1 prio 0 rate 10Mbit ceil 10Mbit burst 14704b cburst 14704b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
 lended: 0 borrowed: 0 giants: 0
 tokens: 9191 ctokens: 9191
```

```
filter parent 1: protocol ip pref 1 fw
filter parent 1: protocol ip pref 1 fw handle 0xa classid 1:10
```

Так, что фильтр будет направлять в шейпер только трафик, помеченный маркером 10 (0xa). Весь немеченный трафик будет проходить в обход шейпера, так как в НТВ не указан поток по умолчанию. Метиться будут лишь пакеты, направляемые на виртуальный адрес wskostja:

```
server:~ # iptables -t mangle -L POSTROUTING
Chain POSTROUTING (policy ACCEPT)
target      prot opt source                destination
MARK       all  -- anywhere             192.168.11.2          MARK set 0xa
server:~ #
```

Последняя деталь – эмулятор помех. По логике, если шейпер на исходящем интерфейсе хоста server имитирует сужение канала, то участок с потерями должен находиться за ним. И у нас не остается иного места, где можно внести (или точнее изъять) потери в трафик, кроме входного интерфейса wskostja:

```
wskostja:~ # iptables -L INPUT
Chain INPUT (policy ACCEPT)
target      prot opt source                destination
DROP       all  -- anywhere             192.168.12.2          random 10%
ACCEPT     all  -- anywhere             192.168.12.2
wskostja:~ #
```

Тут проницательный читатель заметит, что не достает еще эмулятора помех для обратного трафика. Но какой может быть обратный трафик у UDP? И, внося еще помехи для пакетов, движущихся в обратную сторону, мы явно ухудшаем характеристики всех TCP соединений, которые принципиально всегда интерактивны, в отличие от UDP. Скажите несправедливо? Ну и пусть! Нам не чего бояться. Сделаем так:

```
wskostja:~ # iptables -n -L OUTPUT
Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
DROP       all  -- 192.168.12.2         0.0.0.0/0             random 10%
ACCEPT     all  -- 192.168.12.2         0.0.0.0/0
wskostja:~ #
```

Чуть выше уже было замечено, что это очень грубая имитация линии с потерями. Дело в том, что потери являются функцией линии передачи, а не числа переданных пакетов, как в нашем случае. Но даже такая модель позволит понять, что же происходит с трафиком в ненадежных сетях.

Теперь все окружение построено. Договоримся о точках, где будут сниматься данные. В условиях эксперимента будет тип трафика, тип туннеля, объем отправленных с wstovert данных. Скорость передачи нам сообщит команда dd на wstovert. Объем данных поступивших в туннель и число уничтоженных на входе пакетов узнаем через ifconfig tun0 на wsalekseybb. Статистика шейпера на server покажет, сколько данных прошло через ограничитель трафика, сколько пакетов, сколько пакетов было уничтожено. На счетчиках iptables на wskostja узнаем, сколько данных добралось до назначения и сколько было уничтожено, как имитация потерь в линии связи. И для UDP после отключения слушателя netcat получим объем реально полученных данных, подсчитанный командой wc.

С примерами некоторых скриптов, использованных для настроек, можно ознакомиться в архиве [9].

Передача TCP в туннелях.

Сначала проверим самую главную версию, что трафик TCP внутри TCP туннеля якобы плохо передается, а вот внутри туннеля UDP напротив очень хорошо. Для этого поднимем туннель OpenVPN сначала в режиме TCP, затем в режиме UDP, и в каждом состоянии проведем тестовую пересылку без потерь, с потерями 10% и с двухсторонними потерями. Полученные числа соберем в таблицу. К сожалению, так как используется высокоскоростная сеть, тестовый файл не велик, а время не позволяет делать сотни прогонов и затем производить сглаживание результатов, то в таблицу будут занесены наиболее показательные данные из 2-3 прогонов на каждом условии и дополнительно указан возможный разброс.

№№	протокол передачи данных	объем отправленных в Мбайт	режим туннеля	передано в tun0 Мбайт	скорость передачи Мбайт/сек	шейпер, передано Мбайт	шейпер, передано пакетов	шейпер, число задержек	потери %	идеальный трафик шейпера / трафик шейпера
№№	1	2	3	4	5	6	7	8	10	11
1	TCP	105	TCP	103.9	1.2	112	81218	100951	0	1
2	TCP	105	TCP	103.9	0.177 (до 0.188)	128	152585	5504	10	1.14
3	TCP	105	TCP	117.2	0.0736	130	157852	4318	10+10	1.16
4	TCP	105	UDP	103.9	1.2	112	144877	144581	0	1
5	TCP	105	UDP	103.9	0.18 (до 0.19)	124	161677	7995	10	1.11
6	TCP	105	UDP	117.1	0.0791	126	164227	6281	10+10	1.13

Таблица 1. Передача TCP трафика в туннелях.

Колонка 5 не выявила никакого преимущества UDP туннелей в скорости. Средний разброс показателей +/-5%. То есть независимо от несущего протокола внутренний TCP рано или поздно подстраивался под условия передачи и находил примерно одинаковый максимум. А вот показатели колонки 6 имеет более интересную интерпретацию. Итак, в идеальных условиях на шейпере насчитано 112 Мбайт, как в TCP туннеле, так и в туннеле UDP. Но, как только включаем уничтожение 10% трафика, туннель UDP ведет себя пропорционально потерям и добавляет уничтоженное повторной пересылкой дополнительных 10% объема $112 * 1,1 = 123,2$, что примерно равно полученным 124 Мбайт на счетчике шейпера (все индексы в колонке 11). Туннель TCP реагирует на уничтожение трафика предсказанным способом - выполняет дублирование повторов и трафик на шейпере возрастает, но, как это тоже было предсказано, не на полные 20%, а лишь на 14%. Как только включаем уничтожение обратного трафика, то потери еще выше, но опять не 40%, как при полных повторах, и не 20%, а всего лишь 16%! Если сопоставить показатели трафика через туннельный интерфейс (колонка 4) и через шейпер (колонка 6), то использование TCP транспорта добавляет только 4% к переданному объему по сравнению с применением UDP туннеля. То есть, 10% трафика «честно» пропадает в линии передачи, а погрешности двойного дублирования не превышают 4%. Замечу, туннель достаточно условен, сетка высокоскоростная и расхождения RTT минимальны, что показала и утилита tracroute выше по тексту. И все равно, не получается полного удвоения или учетверения. В условиях реального туннеля, где RTT будут различаться больше, скорости будут ниже, а потери, скорее всего, будут меньше, разница в условиях прохождения TCP трафика через TCP и UDP туннель станет совсем незаметной.

Другими словами, никакого существенного ухудшения TCP внутри TCP в нашем

эксперименте замечено не было. Но, если нет ухудшения при работе через TCP туннель, то быть может надо искать улучшение при работе через UDP туннель? И если, проверка TCP трафика не выявила преимущества одного над другим, то может, стоит проверить, как передается UDP трафик в таких туннелях.

Передача UDP в туннелях.

Передача UDP трафика это весьма условное понятие. Фактически это более похоже на ветер. И поэтому понятие скорость передачи теряет всякий смысл. В тестовых проверках скорость случайным образом определялась в диапазоне от 13 Мбайт/сек до 29 Мбайт/сек. Но если на пути такого «ветра» встречается участок с меньшей полосой пропускания, не достаточной чтобы передать все имитированные пакеты, то излишки просто уничтожаются. Как показали прогоны трафика через туннель OpenVPN вместе с шейпером, настроенным на большие предельные скорости, чем 10 Мбит/сек, сам туннель является скрытым ограничителем полосы. При настройках согласно, указанным выше, туннель в режиме UDP ограничивает полосу скоростью 5,9 Мбайт/сек, а в режиме TCP даже еще ниже 2,9 Мбайт/сек. Вероятно, это связано с тем, что туннель организован, как программа, работающая не в пространстве ядра, а в пользовательском пространстве. Но, так или иначе, потери UDP трафика начнутся сразу же на входном интерфейсе туннеля, даже если в нем не настроен собственный шейпер. Второй барьер для UDP будет создавать ограничитель трафика, имитирующий «узкое место», взведенный в нашем случае после хоста server. Ну и, наконец, в тестах с потерей трафика дополнительную убыль пакетов UDP обеспечит фильтрация на входе wskostja. Обратный трафик «портить» не будем, так как это не скажется существенно на выводах. Результаты тестов сведены в таблице 2.

№№	протокол передачи данных	объем отправленных в Мбайт	режим туннеля	передано в tun0 Мбайт	потеряно в tun0 пакетов	шейпер, передано Мбайт	шейпер, передано пакетов	шейпер, уничтожено пакетов	потери %	реально прибыло Мбайт	отправлено прибыло
№№	1	2	3	4	5	6	7	8	9	10	11
1	UDP	105	TCP	1.8	75375	1.94	1406	0	0	1.73	1.04
2	UDP	105	TCP	2.1	75195	2.29	1729	0	10	1.867	1.12
3	UDP	105	UDP	5.3	72744	3.5	4490	1611	0	3.125	1.69
4	UDP	105	UDP	8.0	70678	5.02	6441	2611	10	4.08	1.96

Таблица 2. Передача UDP трафика в туннелях.

Судя по колонке 4, неявный шейпинг самого туннеля привел к тому, что в режиме TCP «пролетело» в интерфейс в два раза меньше пакетов, чем в режиме UDP. Ну и как следствие, окончательно попало в точку назначения (колонка 10) меньше в режиме TCP, чем в UDP. Поскольку туннель TCP кроме всего прочего может подстраиваться под реальную пропускную способность туннеля, то в TCP режиме на шейпере после server нет потерь, а в UDP они присутствуют. Но самые интересные результаты дает расчет отношения трафика, реально отправленного через туннель, и того, что был получен в точке назначения. И в том и в другом режиме работы туннеля это соотношения подчиняется правилу, что трафик в проверке работы с потерями выше трафика в идеальном случае примерно на величину потерь. Например, для TCP $1,04 * 1,1 = 1,14$ приблизительно равно 1,12, полученному в эксперименте. Или для UDP $1,69 * 1,1 = 1,86$ приблизительно равно 1,86, что показал тестовый прогон. Но вот само базовое соотношение пропорции доставки неутешительно для UDP. Получается, что в идеальном случае для TCP туннеля доставляются практически все пакеты – коэффициент 1,04, а вот в туннеле UDP лишь не более чем два из трех – коэффициент 1,69. Здесь еще раз видно, что полученные

результаты могут носить лишь оценочный характер. Ясно же, что коэффициент передачи TCP туннеля должен в идеальном случае равняться единице. Но реальные значения счетчиков интерфейсов интерпретируют по разному пакеты, их заголовки и даже константы пересчета байт в килобайты. Однако общий смысл ясен: UDP туннель совсем не способствует полноте доставки данных до получателя. Главная причина в том, что UDP поток не может подстраиваться под фактическую ширину канала по маршруту роутинга. И именно этим объясняется присутствие специальной опции ограничения канала на входе туннеля OpenVPN. В настоящих настройках она не применялась, а в реальной жизни ее использование позволит сократить расходы на оплату трафика, который в противном случае будет «порезан» одним из маршрутизаторов уже после того, как «провернутся» счетчики ISP. Обращаю внимание, в OpenVPN такая опция есть, а вот в CIPE отсутствует, что вообще не свидетельствует в пользу этого проекта.

Итак, трафик UDP тоже не приобретает ничего хорошего от завертывания его в туннель UDP. Быть может, никакого особого улучшения связи из-за применения CIPE и не происходило? Быть может все это лишь проказы шаловливого кабельщика? Сейчас трудно разобраться, но безымянного кабельщика надо поблагодарить. Ведь именно из-за его работы Олаф Титц, как Колумб, «поплыл в Индию, а прибыл в Америку», то есть неважно, о чем он там думал, когда создавал CIPE, но, безусловно, что его инициатива подтолкнула многих других разработчиков сходных программных средств.

Практические выводы.

Без этой части статья не будет выглядеть завершенной. Если такие выводы были у Олафа, то почему бы и здесь им не появиться. Поскольку в процессе практической проверки никаких особых выгод от использования UDP туннелей не замечено, то вопрос, что использовать переходит в плоскость экспертных оценок. Перечислю их, и пусть читатели сами решат, подходят ли им такие рекомендации.

Во-первых, современные сети являются в большей степени TCP сетями. Они приспособлены для транспортировки именно такого трафика. Системы ограничения полос взаимодействуют с потоками TCP трафика и путем накопления и торможения сегментов в очередях заставляют отправителей TCP замедлять трафик без потерь. Технология предотвращения перегрузки RED (Random early detection) работает только в отношении TCP потоков. Разрабатываются дополнительные функциональные расширения, ориентированные на управление именно TCP трафиком в первую очередь. Например, явное уведомление о перегрузке, или ECN, предложенное в RFC 3168 [10], реализовано как расширение формата TCP. И в том же RFC более половины его содержания посвящено проблеме, как транзитный маршрутизатор сможет управлять с помощью ECN трафиком, завернутым в IPsec. Отдельно там же сказаны и «добрые» слова в отношении других не-TCP туннелей. Про UDP туннели слов нет, поскольку UDP, как достойный транспорт, не рассматривается вообще. Конечно, если очень озаботится, то в Интернете можно разыскать документ «A proposal for the use of ECN bits with UDP flows». Но предлагаю, попробовать найти зарегистрированный RFC на эту же тему. Или попытаться его дождаться.

Во-вторых, если в Вашей сети ситуация «не на высоте», и вопрос стоит не о соответствии последним технологическим новациям, а просто об удовлетворительном уровне работы, то и здесь следует предпочесть TCP транспорт. Очень часто проблемы сетевого взаимодействия связаны с потерями UDP пакетов. Например, замедленный резолвинг сетевых адресов может доставить много неприятностей. На ненадежных подключениях к

Интернету браузеры могут или не открывать запрошенные ресурсы или делать это лишь со второго раза. И учитывая, что UDP трафик самый «беззащитный», то даже на проводных сетях в условиях сильной загруженности будет происходить уничтожение UDP сообщений в первую очередь. Завернув UDP трафик в TCP туннель можно сделать его более надежным и предсказуемым. И часы у Ваших компьютеров будут корректироваться чаще, и DNS будет работать стабильнее. А если в такой туннель завернуть ICMP, то и мониторинг внешних ресурсов также будет стабильнее.

Нет, решительно невозможно обнаружить никаких аргументов против туннелирования в TCP. Быть может, на дату написания статьи [1] все обстояло иначе? Ну, разве что RFC 3168 было в стадии утверждения. А все остальные механизмы регулирования TCP были реализованы задолго до рождения столь оригинальной мысли о неприемлемости туннелирования трафика внутри TCP. Конечно, не стоит забывать о таком эффекте, как вытеснение TCP потока (TCP-starvation/UDP-dominance) именно за счет его управляемости. То есть, если два типа трафика смешиваются в одной полосе регулирования, то TCP трафик, поддающийся регулируемому воздействию, будет уступать полосу UDP. Быть может это и послужило причиной того, что TCP туннель у Олафа Титца не работал, как надо. Но тогда, кроме признания известного недружелюбного кабельщика, надо согласиться с существованием нерадивого сотрудника ISP. Мне кажется, это уже слишком.

Ссылки к статье.

- 1.Olaf Titz. Why TCP Over TCP Is A Bad Idea
<http://sites.inka.de/sites/bigred/devel/tcp-tcp.html>
- 2.Олаф Титц. Почему TCP поверх TCP плохая идея. Перевод А. Барабанова.
http://www.barabanov.ru/arts/tcp_over_tcp/Tcp_over_tcp_rusint.pdf
- 3.RFC 2001
<http://www.protocols.ru/files/RFC/rfc2001.pdf>
- 4.RFC 793
<http://www.protocols.ru/files/RFC/rfc793.pdf>
- 5.Персональная страница Олафа Титца
<http://sites.inka.de/bigred/index.html>
- 6.Домашняя страница CIPE
<http://sites.inka.de/~bigred/devel/cipe.html>
- 7.Домашняя страница OpenVPN
<http://openvpn.net/>
- 8.Сравнительная таблица VPN.
<http://mia.ece.uic.edu/~papers/volans/table.html>
- 9.Примеры скриптов.
<http://www.barabanov.ru/arts/tcp/tovert.tgz>
- 10.RFC 3168
<http://sysadmins.ru/rfc/rfc3168.html>