

Балансировка исходящего веб-трафика между двумя провайдерами для небольших и средних организаций Squid+ipvs+keepalived

Кирилл Хорошилов

Несколько каналов выхода в Интернет уже не редкость для организаций. Как использовать их более эффективно? Решение есть!

В наше время многие компании имеют более одного подключения к Интернету. Как правило, один канал используется как основной, второй – как резервный (по крайней мере, для определенных пользователей либо определенного типа трафика). Очень часто переключение трафика на другой канал при падении первого реализуется администратором вручную.

В статье предлагается способ одновременного использования двух каналов выхода в Интернет с балансировкой нагрузки между ними, автоматическим контролем работоспособности каналов (Dead Gateway Detection). Решение полностью основано на Open Source программном обеспечении.

Вводная часть. Исходная точка

Нужно оговорить ситуацию, о которой пойдет речь. В данном случае мы говорим о так называемом Multihomed подключении локальной сети к двум ISP, когда каждый провайдер выделяет свой IP (или диапазон IP) для подключения. То есть у организации нет собственного диапазона IP-адресов (автономной системы), и протокол ди-

намической маршрутизации (как правило, BGP) не используется. Эта ситуация наиболее часта для небольших и средних организаций.

Наличие двух внешних IP-адресов для исходящего веб-трафика накладывает определенные ограничения для обеспечения сессий веб-приложений (PHP-сессии, cookies). Для поддержания сессионности, которая используется, например, при входе в веб-почту, форум, то есть там, где есть авторизация, а также для работы https, нужно на протяжении определенного времени запросы от одного клиента локальной сети отправлять через один и тот же внешний канал (с одного исходящего IP).

Таким образом, мы вынуждены балансировать трафик по клиентам, а не по http-запросам, т.е. привязывать на определенный промежуток времени каждого клиента к определенному ISP. Следовательно, нагрузка начнет равномерно распределяться по каналам только при большом количестве пользователей (хотя бы больше 10).

Оговорюсь, что, если бы не сессионность, можно сделать балансировку и по запросам, когда даже при наличии одного клиента http-запросы

(GET,POST) будут отправляться поочередно через каждого ISP. Тогда как при использовании BGP таких ограничений, связанных с веб-сессиями, нет, т.к. исходящий IP – один при работе через любой из каналов.

Наиболее известный способ и, пожалуй, единственный для балансировки нагрузки между двумя провайдерами – это применение Advanced routing в Linux (<http://gazette.linux.ru.net/rus/articles/lartc/x348.html#LOADBALANCING>). Однако способ не дает возможности поддерживать веб-сессии, что заставляет искать другие решения. Одно из решений предлагается в этой статье.

Часть 1. Как работает балансировщик

Решение базируется на трех Open Source-продуктах (Squid + ipvs + keepalived). В качестве базовой ОС используется Linux с ядром 2.6. Для определенности будем рассматривать, как все сделать в Fedora Core 6.

Первый продукт – Squid, наиболее распространенный среди прокси-серверов и известен каждому. Squid имеет широкие возможности настройки, часть которых и будем использовать.

Второй продукт менее извест-

тен – ipvs (<http://www.linuxvirtualserver.org/software/ipvs.html>) и является частью проекта www.linuxvirtualserver.org. Это встроенный в ядро Linux load-balancer, frontend для создания балансирующих нагрузку кластеров под Linux. Продукт, кроме нужных возможностей балансировки, хорош тем, что уже встроен в ядро (начиная с версии 2.2!), стабилен и компилируется в ядре по умолчанию. Для его использования достаточно только установить пакет `ipvsadm` – утилиту для настройки.

Мы также будем использовать `keepalived` – демон, управляющий конфигурированием `ipvs`, и настроим его также для проверки работоспособности каналов.

Как компоненты взаимодействуют друг с другом показано на схеме (см. **рис. 1**).

На Host1 размещены `ipvs`-балансер и демон `keepalived`. Последний настраивает `ipvs` и проверяет доступность каналов. `ipvs` балансирует трафик между хостами со `Squid` проху (Host2, Host3), каждый из которых подключен к определенному ISP.

Для проверки каналов `keepalived` будет выполнять запрос определенной веб-страницы из Интернета на каждый из проху-серверов `Squid` с определенным интервалом. Если страница окажется недоступной при запросе через канал несколько раз подряд, канал отключается. Проверочные запросы через канал при этом продолжают с тем же интервалом. Как только канал снова заработает, он будет возвращен в таблицу балансировки `ipvs`.

Варианты реализации решения

Решение, которое описывает схема (см. **рис. 1**), размещается на трех физических машинах. Это не оптимально с точки зрения использования аппаратных ресурсов, и здесь можно предложить несколько более эффективных вариантов размещения сервисов.

Первый из них – это использование виртуальных машин. Задействовав арсенал VMware или, скажем, Xen, или OpenVZ (последние два с точки зрения экономии ресурсов более предпочтительны), можно разместить `keepalived`, `ipvs` и два `Squid`-сервера на одной физической машине, оставив, скажем, `ipvs` + `keepalived` работать на хостовой машине, а `Squid`-серверы поместить в виртуальные среды, подключив к каждой виртуальной машине интерфейс до провайдера.

Вариант интересен, но имеет минус – привлечение дополнительного ПО виртуализации, что усложняет программную реализацию решения и влечет некоторые потери в производительности.

Вариант номер 2 – переместить один из `Squid`-серверов на машину-балансировщик с `ipvs` (Host1), подключить к ней одного из провайдеров, вторую же машину оставить как есть, и балансировать трафик между локальным `Squid` и `Squid`, стоящим на отдельном сервере. Такая реализация требует две физические машины, но без привлечения дополнительного ПО. Кажется, это решение похоже на компромисс...

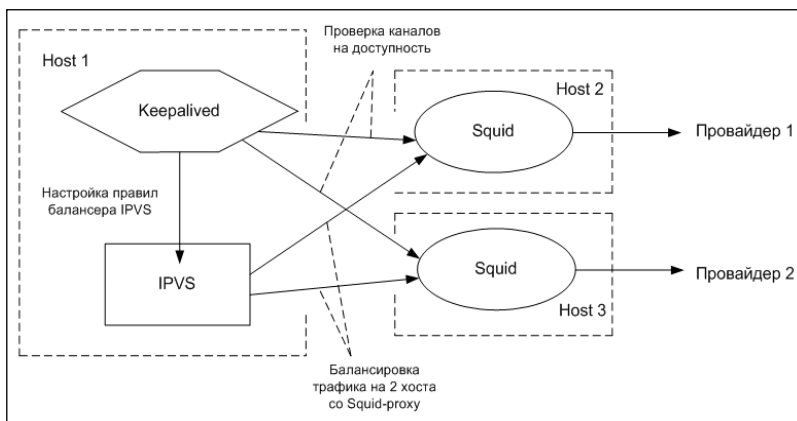


Рисунок 1. Исходная схема взаимодействия компонентов системы балансировки

Третий вариант – переместить `Squid` с обеих (Host2, Host3) машин на Host1 и, воспользовавшись специфичной настройкой `Squid`, заставить его отправлять запросы через разные каналы в зависимости от IP-адреса, на который обращаются клиенты. Однако настройками `Squid` тут не обойтись. «Экономия на железе» требует жертв – необходима небольшая правка ядра. Плюс к тому потребуются настроить правила `source routing`.

Этот вариант решения сложен с точки зрения реализации, но эффективен с позиции оптимального использования аппаратных мощностей. Нужен всего один сервер!

На схеме этот вариант выглядит так (см. **рис. 2**).

Рассмотрим эту реализацию более подробно. Для понимания схема сразу приведена с IP-адресами (взяты произвольно).

Балансировщик `ipvs` принимает запросы на Virtual IP 192.168.1.1 и «разбрасывает», используя NAT-запросы по алгоритму Round Robin (есть и другие алгоритмы балансировки – смотрите `man ipvsadm`), между своими же интерфейсами `eth0` и `eth0:0`, которые слушает `Squid`.

Запросы браузеров, приходящие на `eth0` `Squid`, отправляет в Интернет с адреса интерфейса `eth1`, а запросы, приходящие на `eth0:0`, с адреса интерфейса `eth2` (IP-адрес исходящих пакетов можно задать опцией `tcp_outgoing_address` в конфигурационном файле `squid.conf`). Далее, дописав правила `src routing`'а, мы заставим пакеты, идущие с IP-адресов интерфейсов `eth1` и `eth2`, уходить соответственно через ISP1 и ISP2.

Предлагаю вам определить наиболее подходящий вариант размещения сервисов самостоятельно (или придумать какой-то свой). Каждый из представленных вариантов имеет свои плюсы и минусы. Мне же представляется

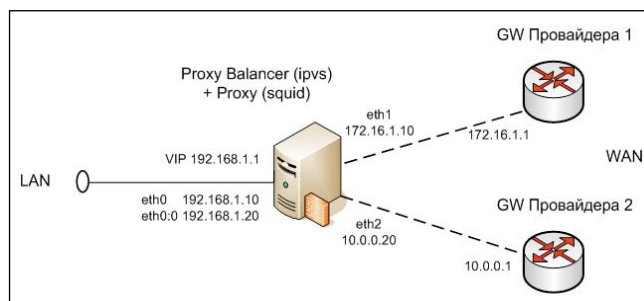


Рисунок 2. Схема балансировки при использовании одного физического сервера

интересным третий вариант. Он наиболее оптимален по использованию аппаратных ресурсов и, как мне кажется, более красив, т.к. требует некоторой изощренной настройки служб и ОС. Поэтому дальше предлагаю перейти к практике и рассмотреть последовательность реализации варианта номер 3 (см. **рис. 2**).

Часть 2. Настройка системы

Для установки системы потребуется сделать следующее:

- найти и установить необходимое ПО;
- пропатчить ядро и установить его в систему;
- сконфигурировать сетевые интерфейсы маршрутизатора;
- настроить squid;
- прописать policy routing;
- настроить keepalived.

Поиск и установка необходимого ПО. Патчинг ядра

Итак, приступим к работе. Начнем с того, что установим необходимое ПО. Ищем пакеты.

Squid, ipvsadm есть в дистрибутиве Fedora. Готовый rpm-пакет keepalived можно взять в репозитории fedora-core-extras (например, тут – <http://mirror.yandex.ru/fedora/linux/extras>). Устанавливаем скачанные rpm:

```
# rpm -ihv squid-2.6.STABLE4-1.fc6.i386.rpm
# ipvsadm-1.24-8.1.i386.rpm
# keepalived-1.1.13-6.fc6.i386.rpm
```

Для создания policy routing также потребуется пакет iproute2. Если в системе нет команды «ip» (/sbin/ip), установите пакет iproute2 из дистрибутива.

Далее нужно пропатчить ядро. Дело в том, что по умолчанию ipvs не поддерживает два узла кластера на одной физической машине, более подробно можно прочитать здесь – <http://archive.linuxvirtualserver.org/html/lvs-users/2005-06/msg00102.html>. Там же внизу страницы – патч от разработчика ipvs – товарища Norms, исправляющий эту особенность ipvs.

Соблюдая правила хорошего тона, соберем rpm с новым пропатченным ядром и потом установим его. Качаем пакет с исходниками ядра: kernel-2.6.22.5-49.fc6.src.rpm и устанавливаем его:

```
# rpm -ihv kernel-2.6.22.5-49.fc6.src.rpm
```

Теперь исходники ядра лежат в /usr/src/redhat/SOURCES.

Скопируем патч с сайта и поместим его в директорию с исходными текстами ядра, как linux-2.6-ipvs.patch. Пропишем в /usr/src/redhat/SPECS/kernel-2.6.spec в соответствующих местах строки совершенно по аналогии, как для других патчей:

```
-----kernel-2.6.spec-----
...
Patch1400: linux-2.6-ipvs.patch
...
ApplyPatch linux-2.6-ipvs.patch
...
```

чтобы скачанный патч применялся к исходникам ядра перед компиляцией.

Еще я отключил функцию проверки GPG-сигнатур для модулей ядра, т.к. мой тестовый сервер не смог набрать необходимую энтропию для генерации ключа GPG, и сборка обрывалась:

```
-----kernel-2.6.spec-----
# Whether or not to gpg sign modules
%define with_modsign 0
```

Все, запускаем сборку rpm ядра.

```
# cd /usr/src/redhat/SPECS
# rpmbuild -bb --target=i686 --with baseonly kernel-2.6.spec
```

Как правило, компиляция не обходится без сюрпризов, но в данном случае у меня все прошло без ошибок.

Если вы используете еще какие-либо настройки при компиляции ядра, не забудьте перенести их в новое ядро. Вручную или используя «make oldconfig/cloneconfig» – удобным для вас методом. Конфигурационный файл, который будет использоваться при сборке, находится в директории с исходными текстами /usr/src/redhat/SOURCES и называется (для архитектуры x86 и опции сборки --target=i686) kernel-2.6.22.5-i686.config.

Инсталлируем готовое ядро:

```
# cd /usr/src/redhat/RPMS/i386
# rpm -ihv kernel-2.6.22.5-49.i386.rpm
```

Отлично. Перезагружаемся с новым ядром.

Замечу здесь, что по-хорошему сборку rpm не следует проводить из-под root. Сборка под root приведена только для упрощения описания (хотя и меня никогда еще не подводила). Root по идее нужен только при инсталляции.

Конфигурация сетевых интерфейсов

Предположим, что шлюз в Интернет на двух провайдеров настроен и работает. Таким образом, имеем 3 интерфейса: первый – eth0 – смотрит в LAN, eth1 и eth2 смотрят соответственно на провайдеров. Предположим, что eth0 имеет IP-адрес 192.168.1.10, eth1 – 172.16.1.10, eth2 – 10.0.0.20.

Squid потребуется 2 внутренних адреса, поэтому поднимем alias eth0:0 на внутреннем интерфейсе:

```
# ifconfig eth0:0 192.168.1.20 netmask 255.255.255.0
```

не забыв прописать это в конфигурационном файле /etc/sysconfig/network-scripts/ifcfg-eth0:0, чтобы alias поднимался после перезагрузки.

Настройка Squid

Теперь в Squid пропишем acl, таким образом, чтобы запросы, приходящие на 192.168.1.10, уходили с адреса интерфейса eth1 (первый провайдер), а приходящие на 192.168.1.20 уходили от Squid с IP интерфейса eth2 на второго провайдера. Делается это с помощью замечательной опции «tcp_outgoing_address» вот так:

```
-----squid.conf-----
acl lhost1 myhost 192.168.1.10
acl lhost2 myhost 192.168.1.20
tcp_outgoing_address 172.16.1.10 lhost1
tcp_outgoing_address 10.0.0.20 lhost2
```

```
# Допишем еще параметр transparent к директиве http_port
http_port 3128 transparent

# Это добавит к обычному режиму transparent (прозрачный)
# режим, когда Squid проксирует http-трафик, проходящий
# через сервер не заметно для клиентов. Данный режим прокси
# требуется для корректного прохождения запросов
# на доступность каналов от keepalived.

# Так же нужно указать Squid не использовать серверные
# persistent connections (persistent connections -
# постоянные соединения, когда для нескольких http-запросов
# используется одно и то же tcp-соединение). В нашем случае
# опцию отключаем, чтобы squid закрывал socket после
# http-запроса, иначе в случае падения канала Squid
# по-прежнему будет отправлять запросы в уже не работающий
# канал, используя открытые socket на интерфейсе
# этого канала.
server_persistent_connections off
```

Настройка политик маршрутизации

Для того чтобы пакеты с src-адресами eth1 и eth2 уходили через соответствующие им интерфейсы, а не на default router, используем policy routing (как описано в документе Linux Advanced Routing & Traffic Control HOWTO <http://gazette.linux.ru.net/rus/articles/lartc/index.html>).

Создадим две дополнительные таблицы маршрутизации T1, T2:

```
# echo 200 T1 >> /etc/iproute2/rt_tables
# echo 201 T2 >> /etc/iproute2/rt_tables
```

Завернем пакеты с src 172.16.1.10 в таблицу T1, и трафик с src 10.0.0.20 в таблицу T2:

```
# ip rule add from 172.16.1.10 table T1
# ip rule add from 10.0.0.20 table T2
```

Пропишем маршруты по умолчанию для трафика, который попал в таблицы T1,T2:

```
# ip route add default via 172.16.1.1 table T1
# ip route add default via 10.0.0.1 table T2
```

Проверим правила командой:

```
# ip ru sh
```

и маршруты в таблицах командами:

```
# ip ro sh t T1
# ip ro sh t T2
```

Сделаем скрипт из этих 6 команд, srcroute.sh и пропишем его на исполнение при загрузке, например в /etc/rc.local.

Конфигурируем keepalived

Теперь нужно настроить keepalived, чтобы он при старте конфигурировал ipvs-балансировщик и выполнял проверку маршрутов.

Вот готовый конфиг с пояснениями:

```
-----/etc/keepalived/keepalived.conf-----
global_defs {

# Определим параметры e-mail для извещения о работе
# health-checker - это полезно, при падении одного
# из каналов вам придет письмо с новостью об этом
notification_email {
    root@host.domain
}

notification_email from keepalived@yourhost.domain
smtp_server 127.0.0.1
smtp_connect_timeout 30
}
```

```
# Настройки относительно VRRP из дефолтного конфигурационного
# файла следует оставить, они необходимы для работы
# keepalived. Подправим их следующим образом
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.1.1
    }
}

# При старте keepalived поднимет виртуальный интерфейс
# с адресом 192.168.1.1, который не будет виден через
# ifconfig (он виден через команду «ip address show»),
# но будет доступен из сети. Этот адрес нужно указать
# в качестве прокси, это VIRTUAL IP балансировщика,
# с которого он будет разбрасывать трафик на реальные
# адреса, слушаемые Squid

# Далее описывается этот виртуальный сервер и 2 реальных
virtual_server 192.168.1.1 3128 {
    delay_loop 6

# Алгоритм балансировки - round-robin
    lb_algo rr

# Способ форварда пакетов на real-сервера - NAT
    lb_kind NAT
    nat_mask 255.255.255.0

# Здесь задается таймаут (в секундах) работы одного клиента
# через одного и того же провайдера (для поддержки
# веб-сессий). Клиент по истечении заданного здесь времени
# бездействия может быть переключен на другой канал
    persistence_timeout 600
    protocol TCP

# Для проверки работоспособности каналов будем проверять
# доступность http://yandex.ru/white.html через каждый
# канал. Страничка достаточно легкая и доступная.
# Можете прописать какую-нибудь свою
    virtualhost yandex.ru

# Real-server - в нашем случае первый адрес, на котором
# слушает Squid
    real_server 192.168.1.10 3128 {
        weight 1

# Для проверки будем слать HTTP GET-запрос и считать,
# что канал доступен, если через него пришел ответ 200
# от сервера
        HTTP GET {
            url {
                path /white.html

# Проверяем http-код ответа сервера при запросе данной
# страницы, этого достаточно, чтобы убедиться, что канал
# работает
                status_code 200
            }
        }

# Время на установление соединения (сек.)
        connect_timeout 10

# Количество попыток соединиться (сек.)
        nb_get_retry 3

# Соответственно задержка между попытками (сек.)
        delay_before_retry 8

# Возможно, вы захотите использовать другие тайм-ауты
    }

# Аналогично для второго Real-server
    real_server 192.168.1.20 3128 {
        weight 1
    }
}
```



```

HTTP GET {
    url {
        path /white.html
        status_code 200
    }
    connect_timeout 10
    nb_get_retry 3
    delay_before_retry 8
}
}

```

На этом конфигурация системы завершена.

Часть 3. Тестирование и отладка

Запускаем keeplived:

```
# /etc/init.d/keepalived start
```

проверяем командой `ipvsadm`, что `ipvs` настроен, каналы проверяются и доступны:

```
# ipvsadm -L
```

```

IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 192.168.1.1:3128 rr persistent 600
-> 192.168.0.10:3128 NAT 1 1 2
-> 192.168.0.20:3128 NAT 1 0 0

```

Так же нужно смотреть `/var/log/messages` на предмет возможных ошибок и запускать `tcpdump` на интерфейсах `eth1` и `eth2` – идет ли прозвон каналов.

Если все отлично, то можно попробовать с клиента настроиться на прокси `192.168.1.1` и посмотреть, пройдет ли запрос. Потом с помощью «`ipvsadm -L`» проверить, на какой из IP-адресов, слушаемых Squid, он сбалансировался (графы в выводе `ActiveConn/InActConn` – см. ранее).

Далее следует убедиться, что пакеты уходят от Squid в нужных направлениях. Делается это, как вы догадываетесь, все тем же `tcpdump` на внешних интерфейсах `eth1` и `eth2` роутера.

Что ж, если все работает, можно попробовать сделать запросы с разных IP, дабы убедиться, что балансировщик распределяет нагрузку.

Дальше подключаем тестовых клиентов и наблюдаем за ситуацией с помощью все тех же средств диагностики: `ipvsadm`, `tcpdump`, `/var/log/squid/cache.log` (добавив, если нужно, детализации в `squid.conf`) и `/var/log/messages`.

Статистику балансировки можно получить командой:

```
# ipvsadm -L --stats
```

```

IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Conns InPkts OutPkts InBytes OutBytes
-> RemoteAddress:Port
TCP 192.168.1.1:3128 22 292 0 35659 0
-> 192.168.0.10:3128 13 172 0 20266 0
-> 192.168.0.20:3128 9 120 0 15393 0

```

Для более удобного восприятия можно настроить рисование графиков с использованием MRTG или аналогичных программ (http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.monitoring_lvs.html).

Выводы

В ситуации, когда динамическая маршрутизация отсутствует, существует дмало возможностей по организации балан-

сировки трафика по двум каналам до провайдеров. По крайней мере, мне известно только 2 способа. Первый описан выше, второй – это использование Advanced routing с опцией `nexthop`, но решение не сработает в паре с веб-прокси, т.к. не имеет нужного функционала для поддержки веб-сессий. (Балансируя трафик, покидающий прокси с помощью `advanced routing`, мы уже не имеем представления, с какого клиента этот трафик и соответственно в какой канал его нужно отправлять в течение какого-то времени.) Решение же, описанное в статье, работает с веб-прокси сервером Squid и поддерживает веб-сессии. Тем не менее предложенная реализация имеет некоторые недостатки. К примеру:

- определенная сложность реализации, требующая перекompilляции ядра;
- балансировка только по клиентам, а не по `http`-запросам или `tcp`-пакетам.

Что касается сложности – она, пожалуй, относительно, т.к. есть системы, гораздо более сложные по своей сути.

Внесение изменений в стандартное ядро является определенным минусом, т.к. осложняет или делает невозможным дальнейшие апдейты ядра в автоматическом режиме. С другой стороны, нередко шлюзы в Интернет устанавливаются единожды и, что называется, на века, без каких-либо последующих апдейтов ядра, либо с достаточно редкими апдейтами.

Балансировка по клиентам является действительно недостатком предложенного решения. Нужно заметить, что это ограничение, накладываемое ситуацией, т.е. двумя внешними IP маршрутизатора. Ограничение будет иметь место только в случае малого количества пользователей прокси. При работе примерно более 10 одновременных клиентов трафик по каналам будет распределяться относительно равномерно.

Если же кто-то решит качать большой файл, то он забьет «свой» канал, при этом второй канал до провайдера будет не нагружен – пользователи второго канала не заметят «проседания» Интернета. Если ваши пользователи скачивают большие объемы данных, следует ограничить скорость загрузки по каждому подключению. В Squid это делается посредством `delay pools` (пулы задержки). Как они настраиваются, вы можете найти в Интернете, например, здесь (<http://bog.pp.ru/work/squid.html>), в разделе «Справедливый дележ канала».

У описанного в статье решения есть и преимущество, о котором еще не говорилось. Предлагаемая система позволяет учитывать неравноценность каналов – стоимость, «толщина». Например, если один канал шире, а второй уже и вы хотите, чтобы нагрузка была пропорциональной – указанием параметра `weight` в конфигурационном файле `keepalived` можно задать вес каждого из каналов. (Если нужно распределить трафик 30/70, поставьте `weight` одному реалсерверу Squid, равный 3, и 7 – второму.)

Таким образом, несмотря на некоторые недостатки и сложности, предложенный в статье способ реализации балансировки может оказаться вполне приемлемым вариантом организации подключения к Интернету, особенно в случае дефицита подобных решений. 