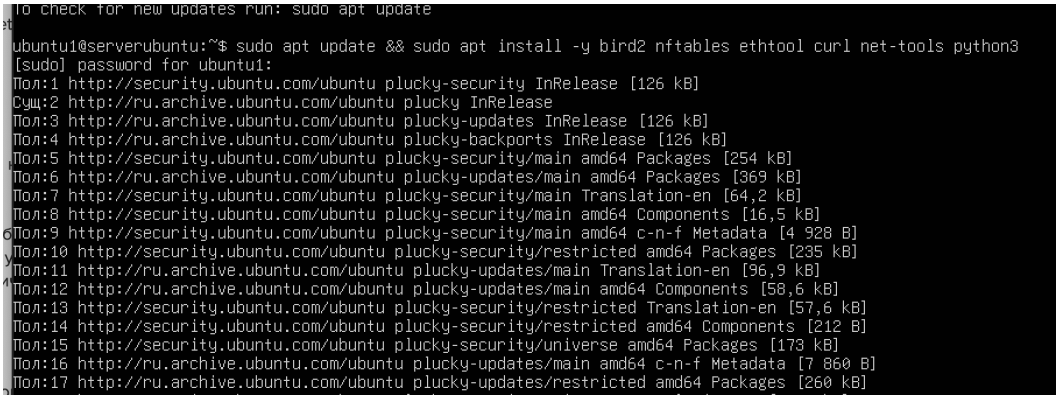
Домашнее задание 3

Сетевой стек

Суханов М.Ю.

Установим все необходимые для работы пакеты:



Устанавливаемые пакеты:

bird2 — BGP/OSPF маршрутизатор.

nftables — современная система файрвола и NAT.

ethtool — инструмент для управления параметрами сетевых интерфейсов.

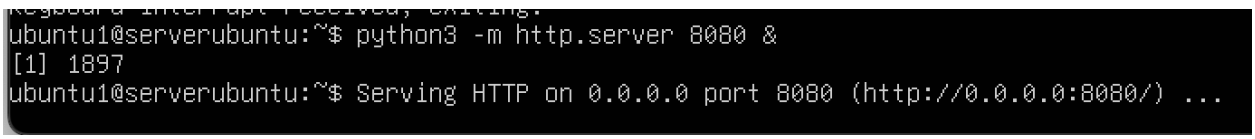
curl — утилита для отправки HTTP-запросов.

net-tools — старые сетевые утилиты (ifconfig, netstat и др.).

python3 — интерпретатор Python.

Задание 1. Анализ состояний TCP-соединений (25 баллов)

1. Запустите Python HTTP сервер на порту 8080:



2. Проверяйте слушающие TCP-сокеты с помощью утилиты ss. найдите сокет с вашим http сервером.

```
ubuntu1@serverubuntu:~$ Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
ss -tlnp | grep 8080
LISTEN 0      5            0.0.0.0:8080      0.0.0.0:*    users:(("python3",pid=1897,fd=3))
ubuntu1@serverubuntu:~$ _
```

Для поиска сокета с нашим http сервером используем команду ss с флагами:

t – TCP, показывает только TCP-сокеты
l – Listen, показывает сокеты на которых сервер принимает новые подключения
n – Numeric, выводит адреса и порты в числовом виде
p – Process, показывает информацию о процессе, который владеет сокетом

Все слушающие TCP-сокеты:

```
ubuntu1@serverubuntu:~$ ss -tlnp
State          Recv-Q         Send-Q                 Local Address:Port                 Peer Address:Port         Process
LISTEN         0              4096                      127.0.0.54:53                      0.0.0.0:*
LISTEN         0              4096                   127.0.0.53%lo:53                      0.0.0.0:*
LISTEN         0              4096                        0.0.0.0:22                      0.0.0.0:*
LISTEN         0              10                        127.0.0.1:9843                    0.0.0.0:*
LISTEN         0              5                           0.0.0.0:8080                    0.0.0.0:*            users:(("python3",pid=1897,fd=3))
LISTEN         0              4096                           [::]:22                         [::]:*
```

3. Подключитесь к серверу через curl.

Для этого будем использовать команду curl с флагом -v (verbose, покажет весь процесс соединения):

```
ubuntu1@serverubuntu:~$ curl -v 0.0.0.0:8080
*   Trying 0.0.0.0:8080...
* Connected to 0.0.0.0 (0.0.0.0) port 8080
* using HTTP/1.x
> GET / HTTP/1.1
> Host: 0.0.0.0:8080
> User-Agent: curl/8.12.1
127.0.0.1 - - [15/Oct/2025 15:45:31] "GET / HTTP/1.1" 200 -
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: SimpleHTTP/0.6 Python/3.13.3
< Date: Wed, 15 Oct 2025 15:45:31 GMT
< Content-type: text/html; charset=utf-8
< Content-Length: 880
<
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a></li>
<li><a href=".bash_logout">.bash_logout</a></li>
<li><a href=".bashrc">.bashrc</a></li>
<li><a href=".cache/">.cache/</a></li>
<li><a href=".lesshst">.lesshst</a></li>
<li><a href=".profile">.profile</a></li>
<li><a href=".ssh/">.ssh/</a></li>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a></li>
<li><a href=".viminfo">.viminfo</a></li>
<li><a href="chared/">chared/</a></li>
<li><a href="homework_key">homework_key</a></li>
<li><a href="python_scrypt.py">python_scrypt.py</a></li>
<li><a href="shared/">shared/</a></li>
<li><a href="shm_creator">shm_creator</a></li>
<li><a href="shm_creator.c">shm_creator.c</a></li>
</ul>
<hr>
</body>
</html>
* we are done reading and this is set to close, stop send
* abort upload
* shutting down connection #0
ubuntu1@serverubuntu:~$ _
```

Из логов можно увидеть, что нам удалось успешно подключится к серверу. В качестве ответа мы получили статус – 200, а также HTML-листинг всех файлов в директории.

4. Проанализируйте состояние TCP-сокетов для порта 8080, объясните, почему есть сокет в состоянии TIME-WAIT, его роль и почему его нельзя удалить.

Переподключимся без флага -v и посмотрим результат работы ss. На этот раз передадим в ss флаг -a вместо -l, который означает показать все соединения:

```
ubuntu1@serverubuntu:~$ curl 0.0.0.0:8080
127.0.0.1 - - [15/Oct/2025 15:56:45] "GET / HTTP/1.1" 200 -
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a></li>
<li><a href=".bash_logout">.bash_logout</a></li>
<li><a href=".bashrc">.bashrc</a></li>
<li><a href=".cache/">.cache/</a></li>
<li><a href=".lesshst">.lesshst</a></li>
<li><a href=".profile">.profile</a></li>
<li><a href=".ssh/">.ssh/</a></li>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a></li>
<li><a href=".viminfo">.viminfo</a></li>
<li><a href="chared/">chared/</a></li>
<li><a href="homework_key">homework_key</a></li>
<li><a href="python_scrypt.py">python_scrypt.py</a></li>
<li><a href="shared/">shared/</a></li>
<li><a href="shm_creator">shm_creator</a></li>
<li><a href="shm_creator.c">shm_creator.c</a></li>
</ul>
<hr>
</body>
</html>
ubuntu1@serverubuntu:~$ ss -tanp | grep 8080
LISTEN    0      5            0.0.0.0:8080       0.0.0.0:*     users:(("python3",pid=1897,fd=3))
TIME-WAIT 0      0          127.0.0.1:8080     127.0.0.1:41628
TIME-WAIT 0      0          127.0.0.1:8080     127.0.0.1:51574
```

TIME-WAIT — это состояние TCP-сокета после закрытия соединения.

Алгоритм работы curl:

1. Открывает TCP-соединение к 0.0.0.0:8080
2. Отправляет HTTP-запрос (GET /)
3. Получает ответ от сервера
4. Закрывает соединение

curl — **инициатор закрытия соединения** (он клиент). По правилам TCP, сторона, закрывающая соединение первой, переходит в состояние TIME-WAIT. TIME-WAIT — это защитный таймаут после закрытия TCP-соединения. Он длится примерно 60-120 секунд и его основная цель – гарантировать корректное завершение TCP-соединения и предотвратить попадание устаревших (запаздывающих) пакетов в новые соединения с теми же IP и портами. Основные проблемы большого количества TIME-WAIT сокетов –

излишнее потребление системных ресурсов, а также возможные проблемы с повторным использованием портов (новые соединения не смогут открываться, так как система не сможет выделить порт).

Задание 2. Динамическая маршрутизация с BIRD (35 баллов)

1. Создайте dummy-интерфейс с адресом 192.168.14.88/32, назовите его service_0.

```
ubuntu1@serverubuntu:~$ sudo ip link add service_0 type dummy
[sudo] password for ubuntu1:
ubuntu1@serverubuntu:~$ ip link show service_0
3: service_0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether aa:b1:e9:ea:46:5c brd ff:ff:ff:ff:ff:ff
ubuntu1@serverubuntu:~$ _
```

Для создания интерфейса использовали команду ip link. Теперь назначим поднятому интерфейсу ip-адрес, поднимем его и просмотрим результат:

```
ubuntu1@serverubuntu:~$ sudo ip addr add 192.168.14.88/32 dev service_0
ubuntu1@serverubuntu:~$ sudo ip link set dev service_0 up
ubuntu1@serverubuntu:~$ ip addr show service_0
3: service_0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether aa:b1:e9:ea:46:5c brd ff:ff:ff:ff:ff:ff
    inet 192.168.14.88/32 scope global service_0
       valid_lft forever preferred_lft forever
    inet6 fe80::a8b1:e9ff:feea:465c/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
ubuntu1@serverubuntu:~$
```

Через ip addr add назначили интерфейсу ip-адрес и через ip link подняли нужный device (dev).

2. При помощи BIRD проаннонсируйте этот адрес при помощи протокола RIP v2 включенного на вашем интерфейсе (eth0/ens33), а так же любой другой будущий адрес из подсети 192.168.14.0/24 но только если у него будет маска подсети /32 и имя будет начинаться на service_.

Анонсирование IP через RIP нужно, чтобы другие маршрутизаторы и устройства в сети знали, через кого достигнуть этот IP. Без анонса адрес будет доступен только локально. Для этого необходимо настроить конфиг bird, который находится по пути “/etc/bird/bird.conf”:

```
# Service routing filter
filter service_filter {
    if (net = 192.168.14.88/32) then {
        accept;
    }

    if (net ~ 192.168.14.0/24 && net.len = 32 && ifname ~ "^service_") then {
        accept;
    }
    reject;
}

# RIP protocol configuration
protocol rip my_rip {
    ipv4 {
        import all;
        export filter service_filter;
    };

    interface "enp0s1" {
        mode broadcast;
        version 2;
        authentication none;
    };
}

# Static route for service_0
protocol static static_service0 {
    ipv4;
    route 192.168.14.88/32 via "enp0s1";
}
```

В конфигурации мы выполнили следующее:

1. Определили, какие маршруты будут анонсироваться через RIP (service_filter). Будут приниматься только маршруты из подсети 192.168.14.0/24 с маской /32 и маршруты по ip-адресу 192.168.14.88/32.

2. Включили протокол RIP v2 на интерфейсе enp0s1 (protocol rip my_rip). Анонсировать будем только маршруты, прошедшие фильтр service_filter.

3. Гарантируем, что конкретный адрес 192.168.14.88/32 будет анонсирован через RIP, даже если динамических маршрутов еще нет.

Перезапустим bird и проверим его статус:

```
ubuntu1@serverubuntu:~/chared$ sudo systemctl status bird
● bird.service - BIRD Internet Routing Daemon
     Loaded: loaded (/usr/lib/systemd/system/bird.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-10-15 18:41:59 UTC; 18min ago
 Invocation: 8c4a8ab8873346fd8c7044e82219bb13
    Process: 10014 ExecStartPre=/usr/lib/bird/prepare-environment (code=exited, status=0/SUCCESS)
    Process: 10021 ExecStartPre=/usr/sbin/bird -p (code=exited, status=0/SUCCESS)
   Main PID: 10023 (bird)
      Tasks: 1 (limit: 3931)
     Memory: 1.2M (peak: 1.6M)
        CPU: 1.210s
     CGroup: /system.slice/bird.service
             └─10023 /usr/sbin/bird -f -u bird -g bird

окт 15 18:41:59 serverubuntu systemd[1]: Starting bird.service - BIRD Internet Routing Daemon...
окт 15 18:41:59 serverubuntu systemd[1]: Started bird.service - BIRD Internet Routing Daemon.
окт 15 18:42:00 serverubuntu (bird)[10023]: bird.service: Referenced but unset environment variable evaluates to an empty string: BIRD_ARGS
окт 15 18:42:00 serverubuntu bird[10023]: Chosen router ID 192.168.14.88 according to interface service_0
окт 15 18:42:00 serverubuntu bird[10023]: Started
```

Сервис успешно поднялся, и конфигурация работает корректно.

3. Создайте ещё три интерфейса:

- service_1 192.168.14.1/30
- service_2 192.168.10.4/32
- srv_1 192.168.14.4/32

Создадим интерфейсы:

```
ubuntu1@serverubuntu:~/chared$ sudo ip link add service_1 type dummy
ubuntu1@serverubuntu:~/chared$ sudo ip link add service_2 type dummy
ubuntu1@serverubuntu:~/chared$ sudo ip link add srv_1 type dummy
```

Назначим ip-адреса:

```
ubuntu1@serverubuntu:~/chared$ sudo ip addr add 192.168.14.1/30 dev service_1
ubuntu1@serverubuntu:~/chared$ sudo ip addr add 192.168.10.4/32 dev service_2
ubuntu1@serverubuntu:~/chared$ sudo ip addr add 192.168.14.4/32 dev srv_1
```

Поднимем интерфейсы:

```
ubuntu1@serverubuntu:~/chared$ sudo ip link set service_1 up
ubuntu1@serverubuntu:~/chared$ sudo ip link set service_2 up
ubuntu1@serverubuntu:~/chared$ sudo ip link set srv_1 up
ubuntu1@serverubuntu:~/chared$ ip addr show | grep -E 'service_|srv_'
3: service_0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 192.168.14.88/32 scope global service_0
4: service_1: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 192.168.14.1/30 scope global service_1
5: service_2: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 192.168.10.4/32 scope global service_2
6: srv_1: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 192.168.14.4/32 scope global srv_1
```

Результат поднятия:

```
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether e2:e1:54:32:e4:7b brd ff:ff:ff:ff:ff:ff
    altname enxe2e15432e47b
    inet 192.168.64.13/24 metric 100 brd 192.168.64.255 scope global dynamic enp0s1
       valid_lft 2285sec preferred_lft 2285sec
    inet6 fd77:c4ad:f6e2:9d2f:e0e1:54ff:fe32:e47b/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 2591928sec preferred_lft 604728sec
    inet6 fe80::e0e1:54ff:fe32:e47b/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
3: service_0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether aa:b1:e9:ea:46:5c brd ff:ff:ff:ff:ff:ff
    inet 192.168.14.88/32 scope global service_0
       valid_lft forever preferred_lft forever
    inet6 fe80::a8b1:e9ff:feea:465c/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
4: service_1: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether 06:f7:c6:c9:f4:47 brd ff:ff:ff:ff:ff:ff
    inet 192.168.14.1/30 scope global service_1
       valid_lft forever preferred_lft forever
    inet6 fe80::4f7:c6ff:fec9:f447/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
5: service_2: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether 7e:2d:06:91:b8:46 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.4/32 scope global service_2
       valid_lft forever preferred_lft forever
    inet6 fe80::7c2d:6ff:fe91:b846/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
6: srv_1: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 46:aa:8c:43:8e:c9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.14.4/32 scope global srv_1
       valid_lft forever preferred_lft forever
    inet6 fe80::44aa:8cff:fe43:8ec9/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
```

Выполним проверку того, что анонсируются только нужные адреса. Для этого воспользуемся tcpdump:

```
sudo tcpdump -i enp0s1 -n udp psudo birdc show route export my_rip
BIRD 2.16.1 ready.
Table master4:
192.168.14.88/32     unicast [direct1 19:56:54.535] * (240)
        dev service_0
```

Из результата можно увидеть, что анонсируется только нужный маршрут 192.168.14.88/32. Маршруты service_1, service_2 и srv_1 не анонсируются, так как отсеиваются на уровне фильтра. У srv_1 проблемы с наименованием, у service_1 с маской, а у service_2 неверная подсеть.

Задание 3. Настройка фаервола/ Host Firewalling (25 баллов)

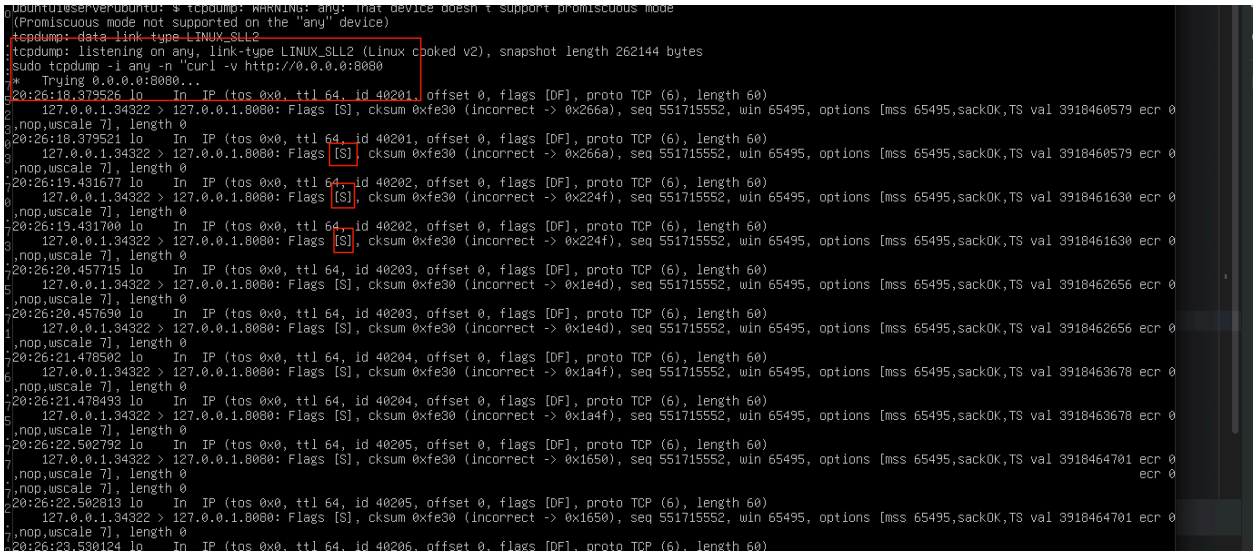
1. С помощью iptables или nftables создайте правило, запрещающее подключения к порту 8080.

```
ubuntu1@serverubuntu:~$ sudo iptables -A INPUT -p tcp --dport 8080 -j DROP
ubuntu1@serverubuntu:~$ sudo iptables -A INPUT -p udp --dport 8080 -j DROP
ubuntu1@serverubuntu:~$ sudo netfilter-persistent save
run-parts: executing /usr/share/netfilter-persistent/plugins.d/15-ip4tables save
run-parts: executing /usr/share/netfilter-persistent/plugins.d/25-ip6tables save
ubuntu1@serverubuntu:~$ sudo netfilter-persistent reload
run-parts: executing /usr/share/netfilter-persistent/plugins.d/15-ip4tables start
run-parts: executing /usr/share/netfilter-persistent/plugins.d/25-ip6tables start
```

Данными командами мы:

- Заблокировали входящие подключения к порту 8080 по TCP
- Заблокировали входящие подключения к порту 8080 по UDP
- Сохранили правила

Проверим работу фаерволла:



```
* connect to 0.0.0.0 port 8080 from 127.0.0.1 port 34322 failed: Connection timed out
* Failed to connect to 0.0.0.0 port 8080 after 133868 ms: Could not connect to server
```

Работа фаерволла проверялась через следующий алгоритм команд:

1. sudo tcpdump -i any -n "port 8080" -v

Через tcpdump запустили мониторинг трафика на порту 8080

2. curl -v http://localhost:8080

Попытались подключиться к серверу

Результатом подключения стали многократные попытки отправки SYN пакетов (попытки установки соединения). Пакетов ответа [S.] не наблюдалось, следует, что порт не доступен.

Задание 4. Аппаратное ускорение сетевого трафика (offloading) (15 баллов)

1. С помощью ethtool исследуйте offload возможности вашего сетевого адаптера.

Исследуем offload возможности основного интерфейса enp0s1:

```
ubuntu1@serverubuntu:~$ ethtool -k enp0s1 | grep tcp-segmentation-offload
tcp-segmentation-offload: on
ubuntu1@serverubuntu:~$ ethtool -k enp0s1 | grep generic-segmentation
generic-segmentation-offload: on
ubuntu1@serverubuntu:~$ ethtool -k enp0s1 | grep generic-receive-offload
generic-receive-offload: on
ubuntu1@serverubuntu:~$ ethtool -k enp0s1 | grep large-receive-offload
large-receive-offload: off [fixed]
ubuntu1@serverubuntu:~$ _
```

1. tcp-segmentation-offload: on – разбиение сетевым адаптером больших TCP-пакетов на более мелкие сегменты

2. Generic Segmentation Offload: on - оптимизация сегментации пакетов на уровне ядра

3. Generic Receive Offload: on – объединение мелких входящих пакетов в более крупные

4. Large Receive Offload: off - более агрессивная технология по сравнению с Generic Receive Offload (выключена).

TCP Segmentation Offload (TSO) решает задачу снижения нагрузки на центральный процессор (CPU) при передаче больших объемов данных по сети за счет переноса работы на специализированный процессор сетевого адаптера.