**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ТАРАСА ШЕВЧЕНКА**
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

# Розгортання інтернет-шлюзу засобами пакету VPP

Кваліфікаційна робота бакалавра
студента 4 року навчання
спеціальність: 123 «Комп'ютерна інженерія»
Івана БУТИЛОВА

_____

Науковий керівник
кан. тех. наук Олександр БОРЕЦЬКИЙ
асистент кафедри комп'ютерної інженерії

_____

Рецензент
кан. фіз.-мат. наук Іван КОЛОМІЄЦЬ
асистент кафедри електрофізики

_____

До захисту допускаю: Завідувач кафедрою

_____Юрій БОЙКО

Ухвалено на засіданні кафедри "____"_____2022 р., протокол № ____

Київ - 2022

# РЕФЕРАТ

Кваліфікаційна робота бакалавра:

Дана робота полягає в пошуку оптимального варіанту конфігурації для інтернет-шлюзу в мережі Білінгової Системи Київського Національного Університету імені Тараса Шевченка, з метою збільшення продуктивності в межах обробки пакетів за протоколом NAT. В процесі роботи було досліджено можливості та реальний приріст продуктивності, що можна досягнути розгортанням шлюзу засобами пакету VPP, в порівнянні зі старою конфігурацією на основі створеної в процесі тестової мережі.

Ключові слова: NAT, Vector Packet Processing, Інтернет-шлюз

**ЗМІСТ**

# ВСТУП

В сучасних реаліях мережі Інтернет ми маємо проблему відсутності достатньої кількості адрес за протоколом IPv4, що обмежує кількість можливих пристроїв в мережі. Повним вирішенням проблеми став протокол IPv6, але до цього часу для вирішення цієї проблеми використовувався протокол NAT. І дотепер він знаходить застосування в мережевій інфраструктурі, оскільки надає досить зручні функції для адміністрування підмережі та загального контролю кібербезпеки в організаціях.

Реалізація протоколу NAT полягає в розділенні інтернет-шлюзом загальної мережі на внутрішню (LAN) та зовнішню (WAN) мережу. В адресному просторі внутрішньої мережі пристрої мають, зазвичай, приватні адреси, що не дозволяють бути частиною мережі інтернету. Для отримання доступу до мережі інтернет, мережевий шлюз виконує трансляцію внутрішньої адреси в зовнішню, або ж публічну, з відповідного пулу доступних адрес. Таким чином за десятками публічних адрес можуть знаходитися сотні або тисячі внутрішніх пристроїв.

# 1. ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ФУНКЦІОНУВАННЯ ІНТЕРНЕТ-ШЛЮЗІВ НА АПАРАТНІЙ ПЛАТФОРМІ X86

## 1.1. Огляд мережі гуртожитків

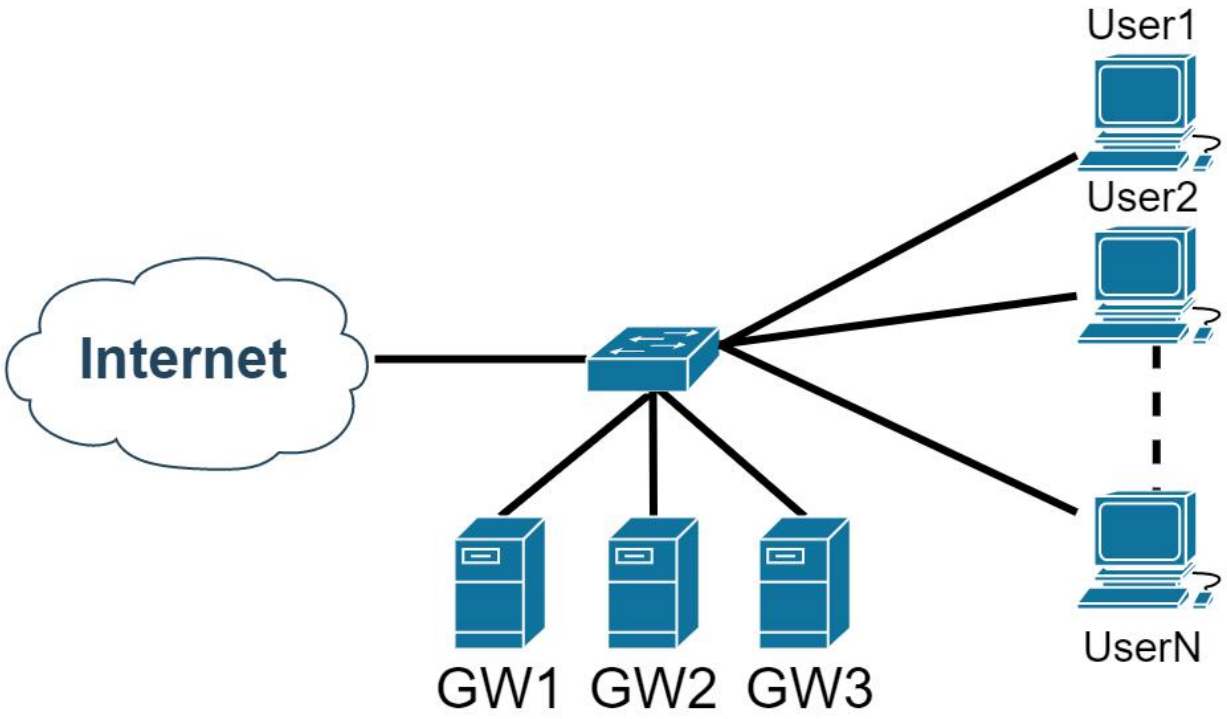


Рис.1

Мережа Білінгової Системи Київського Національного Університету імені Тараса Шевченка виконує завдання надання доступу до мережі інтернет користувачам в об'єднаній мережі гуртожитків. Функціональні обов'язки інтернет-шлюзу на принциповій схемі мережі (рис.1) виконують три сервери-шлюзи: GW1, GW2, GW3. Навантаження між ними розділене комутатором на відповідні частини: GW1 - 25% трафіку, GW2 - 25% трафіку, GW3 - 50% трафіку.

Сервери-шлюзи мають ідентичні конфігурації, тому їх можна розглядати як логічно один мережевий шлюз. В години пік шлюз обробляє вхідний трафік зі швидкістю 1,5 Гбіт/с та обслуговує близько 4000 тисяч користувачів.

## 1.2. Проблема поточної конфігурації

В наслідок розширення внутрішньої мережі, збільшення користувачів та пристроїв, якими вони користуються, відповідно і збільшується об’єм трафіку, що доводиться обробляти мережевому шлюзу. Через обмеженість виділених коштів на підтримку роботи мережі, стає неможливим оновлення апаратної частини мережевих шлюзів. Наявне обладнання можна вважати застарілим, наприклад мережевий шлюз GW3 використовує Mellanox ConnectX-3, що вже досяг стадії End Of Life (EOF), а отже оновлень ПЗ виробник не випускає та обслуговування скорочено до мінімуму. Тому залишається лише збільшення продуктивності за рахунок конфігурування програмної частини , що дозволить використовувати ресурси обладнання ефективніше, а отже збільшити продуктивність.

Також поточна конфігурація не дозволяє впровадити нові функції з метою розвитку мережі. А саме:

1) Створення і менеджмент NAT-сесій на основі MAC-адреси. Це важливо, оскільки при майбутній реалізації IPv6, виникає потреба динамічної маршрутизації публічних адрес IPv6, у випадках коли користувач переноситься на інший шлюз.

2) Обмеження кількості сесій на одного користувача. Є потребою суто адміністративною.

3) Рівномірний та динамічний розподіл сесій по IP-адресам.

## 1.3. Огляд ipfw nat

Поточна конфігурація мережевого шлюза виконана застосовуючи утиліту ipfw. Розглянемо чим воно себе являє та чи є можливість переконфігурувати його впровадивши нові функції.

Ipfw або ipfirewall - це реалізація фаерволу, маршрутизації та NAT в ядрі FreeBSD. Має користувацький інтерфейс, що дозволяє в форматі списку нумерованих правил, оформлювати інструкції для фільтрування, перенаправлення та трансляції пакетів, що надходять до ядра ОС, відповідно йдучи за зростанням номера правила.

Оскільки, не видно шляхів написання нових конфігурацій, що додадуть продуктивності його роботи, розглянемо синтаксичні конструкції що дозволить реалізувати потрібні функції. І тут маємо певне розчарування. Існує фільтрація по MAC-адресу, але нею неможливо прив’язати до сесії. Відповідно і немає менеджменту сесії, часовий контроль здійснюється оновленням списку фільтрування перед інструкціями транслювання.

Окремо розглянемо реалізацію розподілу сесій по IP-адресам. Процес полягає в отримані залишку ділення внутрішньої адреси на 128 та зв’язування її з зовнішньою адресою. Це можна назвати статичним розподілом і доволі не рівномірним, оскільки кількість внутрішніх користувачів з адресами x.x.x.126 менша. Особливих інструкцій для створення динамічного розподілу не існує.

Ipfw не задовольняє висунуті вимоги, тому маємо опрацювати інше рішення. Оскільки не тільки FreeBSD має in-kernel реалізацію мартшрутизації та трансляції, а й ОС на базі Linux, розглянемо виконання NAT на базі фаерволу в ядрі Linux.

## 1.4. Огляд iptables

Iptables - це утиліта в адресному просторі користувача, що дозволяє встановлювати фільтраційні правила для IP пакетів, що проходять фаервол ядра Linux. Правила організовуються в впорядковані ланцюги. Таблиці, що створюються iptables, є контейнерами ланцюгів, які логічно зв’язані з певним процесом обробки пакетів.

Синтаксис iptables дозволяє не тільки транслювати пакети, але й виконувати розподіл, або балансування, NAT-сесій між зовнішніми ip-адресами не тільки статично, але й по алгоритму Round-robin та випадкового вибору. Проте на жаль зберігається проблема, що характеризує фаервольну природу iptables, в присутності фільтрації за MAC-адресою, але відсутності створення та контролю над такою сесією.

Щодо продуктивності, нема чітких досліджень, проте вважається що ядра BSD працюють дещо швидше ядер Linux. Тому без проведення тестування важко судити.

Після розгляду iptables, виникає потреба в розгляді принципово іншого рішення. Відмовитися від ядрового фаерволу та розглянути рішення, що “забирає” контроль над мережевими інтерфейсами у ядра та передає контроль програмам адресного простору користувача, наприклад використовуючи DPDK-бібліотеки.

## 1.5. Огляд VPP

VPP або Vector Packet Processing є модульним фреймворком для обробки стаку TCP/IP протоколів. Є open-source реалізацією VPP-технології американської компанії Cisco.

Згідно назви в основі пакету лежить векторна обробка пакетів. В класичній моделі скалярної обробки відбувається послідовна обробка одного пакету перед тим як почати обробку наступного пакету. На відмінну від скалярної, векторна обробка полягає в формуванні вектору пакетів (рис.12), відповідний пакет проходить аналогічну обробку. Оскільки для обробки однакових пакетів не потрібно перезавантажувати інструкції в кеш, відбувається “розминка” кешу інструкцій і зменшення часу між обробками пакетів. Отже в результаті відбувається значне підвищення кількості оброблених пакетів за однаковий проміжок часу.

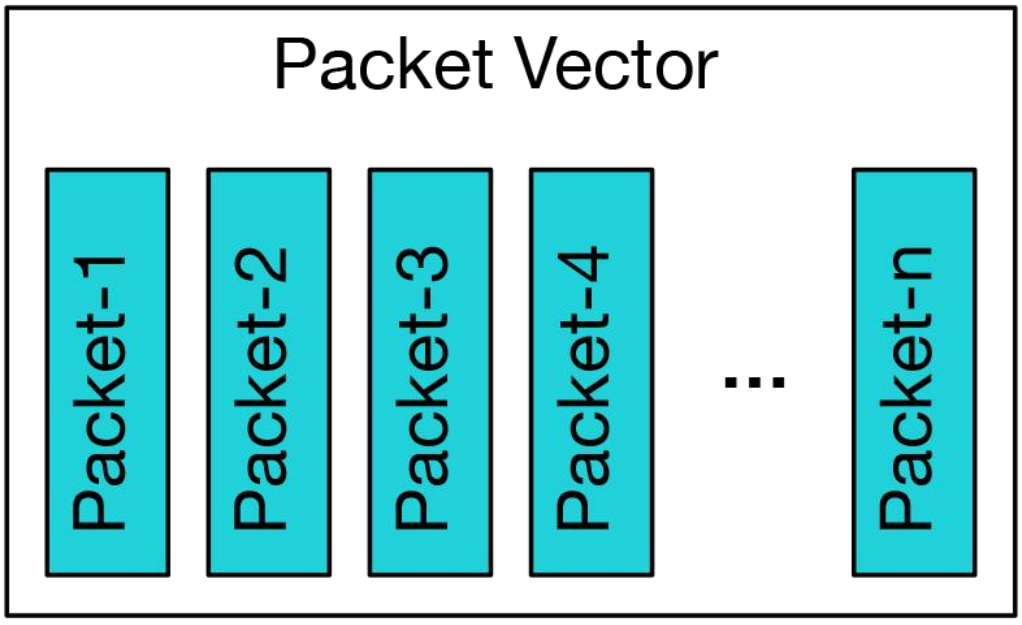


Рис. 2 Вектор мережевих пакетів

Оброблюванні пакети йдуть за процесуальним графом (рис.3). Модульність пакету VPP полягає в легкості модифікування графу за рахунок підключення плагінів (рис.4).

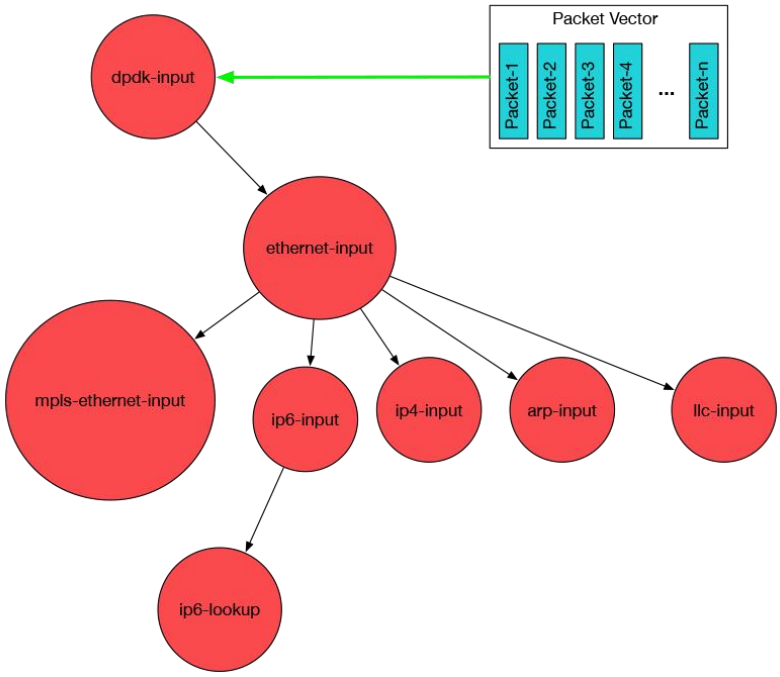


Рис. 3

Плагіни виконують специфічну до задачі обробку пакетів, що дозволяє сконфігурувати розширений функціонал та не перенавантажувати систему. Відповідно можна сконфігурувати систему лише під окремий процесс обробки, відокремивши непотрібні обробки.


Рис. 4

Існують вже готові плагіни, створенні для розгортання основного функціоналу пакету VPP. А саме:

**1. IPv4/IPv6**

*1.1. 14+ MPPS, single core*

*1.2. Multimillion entry FIBs*

*1.3. Input Checks*

*1.3.1. Source RPF*

*1.3.2. TTL expiration*

*1.3.3. Header checksum*

*1.3.4. L2 length < IP length*

*1.3.5. ARP resolution/snooping*

*1.3.6. ARP proxy*

*1.4. Thousands of VRFs*

*1.4.1. Controlled cross-VRF lookups*

*1.5. Multipath – ECMP and Unequal Cost*

*1.6. Multiple million Classifiers –*

*1.6.1. Arbitrary N-tuple*

*1.7. VLAN Support – Single/Double tag*

**2. IPv4**

*2.1. GRE, MPLS-GRE, NSH-GRE,*

*2.2. VXLAN*

*2.3. IPSEC*

2.4. *DHCP client/proxy*

3. **IPv6**

*3.1. Neighbor discovery*

*3.2. Router Advertisement*

*3.3. DHCPv6 Proxy*

*3.4. L2TPv3*

*3.5. Segment Routing*

*3.6. MAP/LW46 – IPv4aas*

*3.7. iOAM*

**4. MPLS**

*4.1. MPLS-o-Ethernet –*

*4.1.1. Deep label stacks supported*

5. **L2**

*5.1. VLAN Support*

*5.1.1. Single/ Double tag*

*5.1.2. L2 forwarding with EFP/BridgeDomain concepts*

*5.2. VTR – push/pop/Translate (1:1,1:2, 2:1,2:2)*

*5.3. Mac Learning – default limit of 50k addresses*

*5.4. Bridging – Split-horizon group support/EFP Filtering*

*5.5. Proxy Arp*

*5.6. Arp termination*

*5.7. IRB – BVI Support with RouterMac assignment*

*5.8. Flooding*

*5.9. Input ACLs*

*5.10. Interface cross-connect*

Однак на вбудованих список плагінів не закінчується, бо пакет VPP дозволяє використовувати власноруч написані плагіни, або звісно плагіни інших розробників, що надали доступ до вихідного коду на загальнодоступних ресурсах, або іншими шляхами. Для створення власного плагіну пакет надає зручне низькорівневе API, що дозволяє на мові C писати та модифікувати існуючі плагіни.

Таким чином за допомотою NAT-плагіну ми можемо можемо виконувати менеджмент сесій, що відповідає нашим вимогам. Та за допомогою mactime-плагіну реалізувати створення сесій за MAC-адресами.

## 1.6. Результати дослідження

На основі оглянутих реалізацій NAT-протоколу для апаратної платформи x86, можна заключити:

1. Ipfw вичерпав свої можливості. Ми не можемо збільшити його продуктивність та реалізувати зазначені функції.

2. Iptables може реалізувати частину функцій, проте без тестування ми не можемо говорити про збільшення продуктивності.

3. VPP має найбільший потенціал, бо за допомогою нього можна реалізувати зазначені функції та завдяки іншому підходу до обробки пакетів, може показати відмінну від поточної конфігурації продуктивність.

Тому виходячи з наших вимог слід використати конфігурацію на основі пакету VPP.

# 2. ІНТЕРНЕТ-ШЛЮЗ НА ОСНОВІ VPP

## 2.1. Архітектура рішення

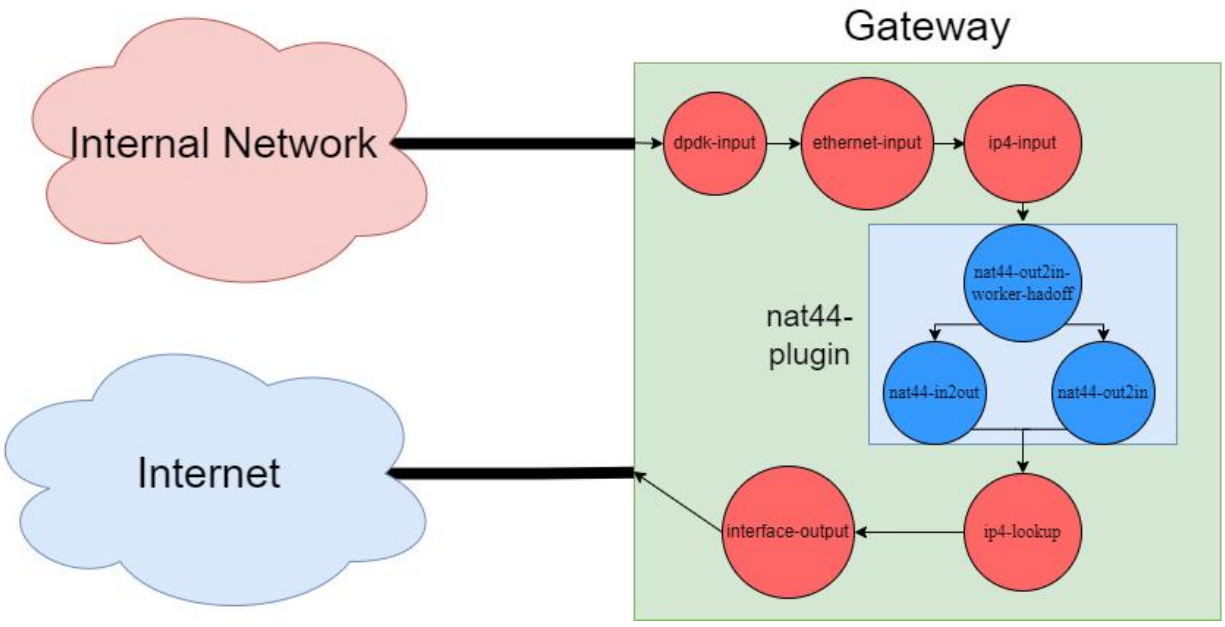


Рис.5

Рух трафіку від користувачів до мережі Інтернет по принциповій схемі (рис.5) буде йти шляхом з мережі гуртожитків до мережевого шлюзу. В мережевому шлюзі пакети трафіку формуються в вектори та проходять низку нодів сервісу VPP. Пройшовши обробку, трансльований відправляється до адресата в мережу інтернет. Рух пакетів із мережі інтернет до користувача пройде шлях в оберненому напрямку.

## 2.2. Граф нодів для пакету IPv4

В середовищі сервісу VPP, пакет проходить низку нодів. Доцільним буде зазначити граф проходження до точки в якій вступає в силу NAT-плагін.

Вхідний нод графа це dpdk-input. Саме тут набір бітів з фізичного інтерфейсу трансформуються в Ethernet фрейми. Обробкою фреймів займається наступний за графом нод ethernet-input. В процесі вони декапсулюються в пакети IPv4. Їх обробкою вже займається нод ip4-input. Наступний після нього йде перший нод NAT-плагіну.

## 2.3. Огляд NAT-плагіну

Пакет VPP має вбудовані NAT-плагіни, тому доцільно обрати потрібний нам. В вихідних кодах є відповідно NAT44, NAT66, NAT64 та PNAT. Оскільки ми будемо працювати з IPv4 адресами на початковому етапі, буде доцільно обрати NAT44, оскільки NAT64 буде працювати для внутрішніх мереж з IPv6 адресацією.

Огляд графу нодів плагіну дає висновок про шлях обробки пакетів (рис.6). Вхідною точкою є нод nat44-in2out-worker-handoff, що отримує пакет з ноду ip4-input. Він визначає індекс відповідальний обчислювального блоку та хешує ip-адресу відправника. Якщо індекс обчислювального блоку збігається з працюючим, то пакет перенаправляється до ноду nat44-in2out, інакше викидує пакет зі свого потоку обробки. Нод nat44-in2out визначає відповідну сесію. Ключем сесії є адреса відправника, та порт з якого надійшов пакет, а також протокол четвертого рівня. Відбувається хешування таблиці сесій. За відсутності існуючої сесії, пакет передається ноду nat44-in2out-slowpath. Якщо сесія існує - відбувається трансляція ip-адреси та порту відправника, оновлення лічильника сесії. Пакет передається наступному ноду, зазвичай ip4-lookup. На ноді nat44-in2out-slowpath відбувається реєстрація сесії в системі. Ключем сесії є адреса відправника, протокол четвертого рівня. Відбувається хешування з таблиці користувачів, за відсутності створюється новий запис. Наступною йде звірка квоти на сесії з поточним значенням лічильника сесій користувача. Створюється

сесія, відповідний маппінг, зазначений в налаштуваннях конфігурації плагіну, та хеш-заголовок для in2out, out2in нодів. В завершення відбувається трансляція пакету та направлення до наступного плагіну.

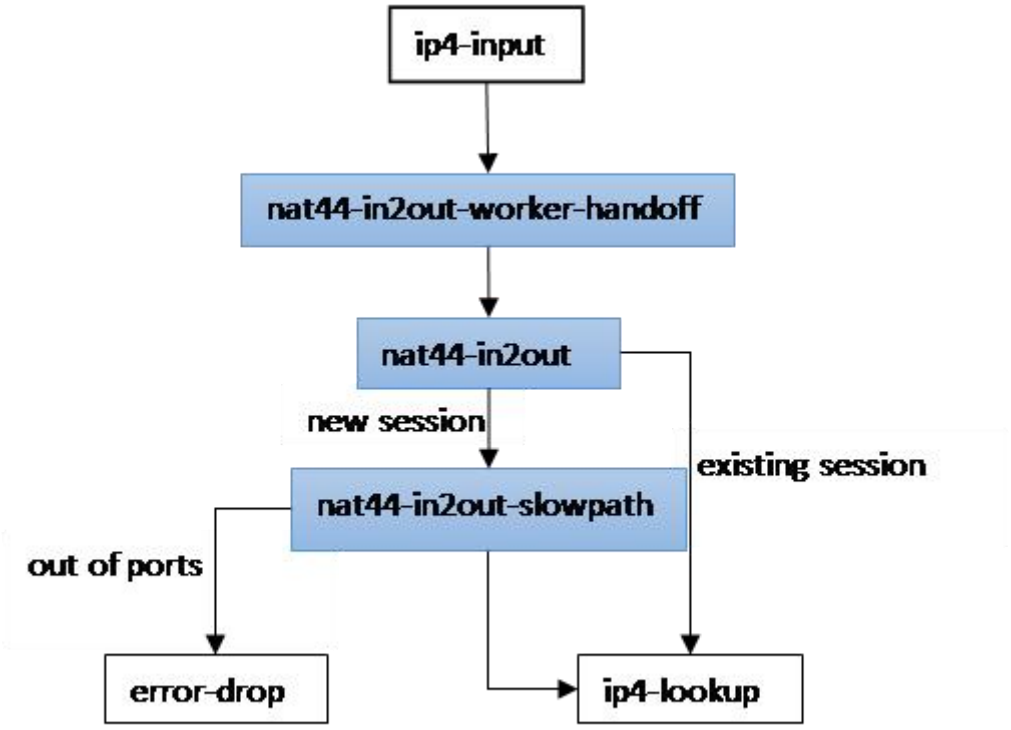


Рис. 6

Оглянувши рух пакету з внутрішньої мережі до зовнішньої, зробимо аналогічний огляд руху в протилежному напрямку (рис.7). Вхідною нодою є nat44-out2in-worker-hadoff, що аналогічно визначає індекс відповідальний обчислювального блоку. Ключем є ip-адреса та порт отримувача. Відповідно якщо індекс обчислювального блоку збігається з працюючим, то пакет перенапрвляється до ноду nat44-out2in, інакше викидує пакет зі свого потоку обробки. На ноді nat44-out2in визначається відповідна сесія, у випадку незнаходження нод спробує підставити до статичного маппінгу. Якщо спроба буде невдалою станеться викид пакету і запис в лог помилки. Опісля трансляція адрес та порту відправника з передачею пакету наступному ноду.

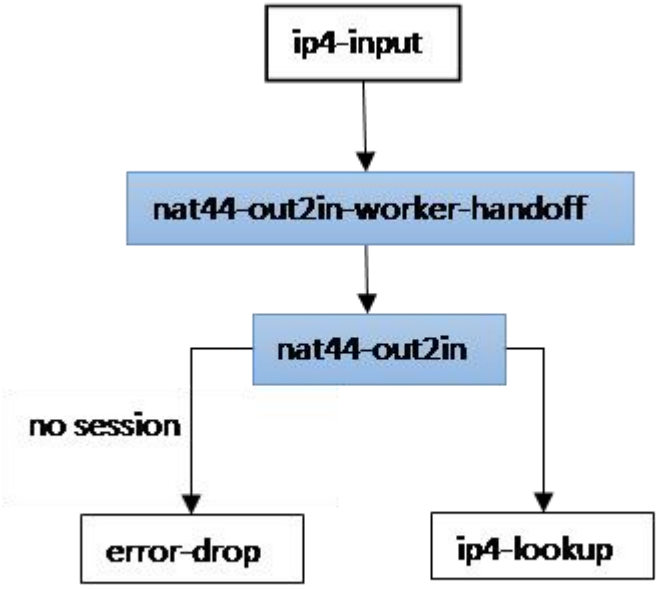


Рис. 7

## 2.4. Огляд плагіну mactime

Помітно, що на схемі архітектурного рішення не вказано ноди плагіну mactime. Це пояснюється тим, що вони викликаються внутрішнім середовищем VPP у випадку їх необхідності, а саме при створені плагіном NAT сесії та перевірки існування інших сесій.

Варто зазначити конфігурування цього плагіну, оскільки він не використовується на тестовому мережевому шлюзі в наступному розділі. Для активації плагіну, достатньо включити його відповідними командами на внутрішніх, стосовно NAT, інтерфейсах

*bin mactime_enable_disable *інтерфейс**

Опісля вимагається створення бази даних заповненої правилами що пропускають чи забороняють сесії для відповідних MAC-адрес з чи без часових обмежень. Варіанти правил:

*allow-static* - пропуск сесії для відповідної MAC-адреси

*drop-static* - заборона сесії для відповідної MAC-адреси

*allow-range* - пропуск сесії для відповідної MAC-адреси в зазаначених часових рамках

*drop-range* - заборона сесії для відповідної MAC-адреси в зазаначених часових рамках

Приклад:

*bin mactime_add_del_range name user_09 mac 00:ab:dc:ef:fe:00 allow-range Mon 7:59 - 18:01*

# 3. РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ІНТЕРНЕТ-ШЛЮЗУ

## 3.1. Інфраструктура тестування

Для тестування зручно спростити мережу до формули домашнього маршрутизатора. Умовно кажучи, наш шлюз буде адмініструвати доступ до мережі невеликій кількості внутрішніх пристроїв. Для своєї внутрішньої мережі я розвернув дві віртуальні машини на основі гіпервізора VMware Workstation 15.

Оскільки я провожу симуляцію на віртуальних машинах, а отже пристрої будуть використовувати ресурси мого персонального комп'ютера, що зумовить деякі поправки до параметрів генерації трафіку. А саме час та використання ядер у TRex сервера. Найбільш доцільним я бачу проведення тестування в 15 хвилин, оскільки разом з шлюзом під тестом будуть зайняті всі ядра мого процесору, а отже можливі затримки на обслуговування викликів моєї домашньої операційної системи. Також і обмеження в одне ядро для Trex серверу з тих же причин.

Для опису використання ресурсів я склав таблицю:

| Кількість | VPP шлюз | TRex сервер | Наявних в ПК |
|---|---|---|---|
| Виділеної оперативної пам'яті | 8 Gb | 4 Gb | 16 Gb |
| Виділених ядер процесору | 4 ядра | 2 ядра | 6 ядер |

Схематично тестування відбуватиметься за Stateful моделлю (рис.6), тобто генератор трафіку знаходиться на окремому сервері (TRex Server), що з'єднаний двома мережами з шлюзом VPP під тестуванням (DUT).

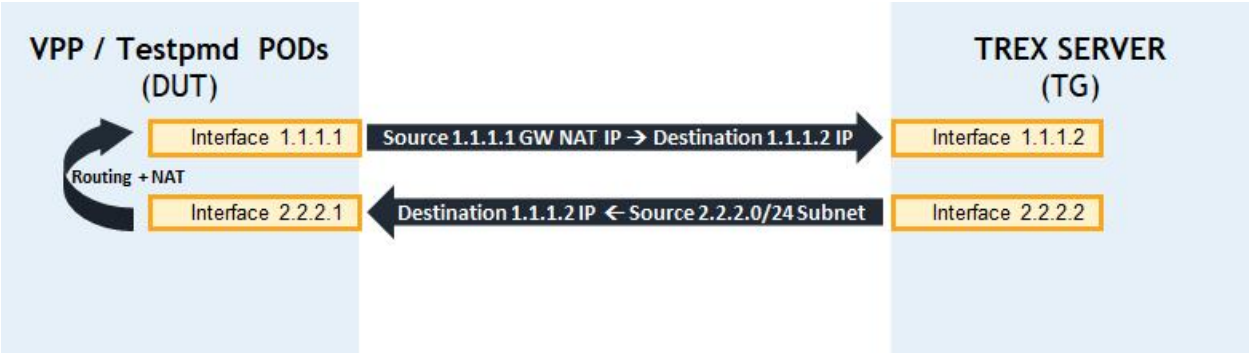


Рис. 8

### 3.2. Встановлення та налаштування інтернет-шлюзу

Оскільки ipfw та iptables є в стандартному пакеті своїх операційних систем, то їх встановлення ми не розглядаємо. Ситуація з VPP ускладнюється відсутністю його в пакетних менеджерах. Тому доводиться вручну прописувати шлях до репозиторію fd.io, в якому містяться пакети релізних та розробницьких версій VPP.

*curl -s https://packagecloud.io/install/repositories/fdio/release/script.rpm.sh | sudo bash*

Після цього ми можемо з впевненістю завантажувати залежності для VPP:

*yum install epel-release mbedtls python36*

Виконавши інсталяцію залежностей, можна переходити до пакету VPP:

*sudo apt-get install vpp*

Опісля встановлення можна зробити тестовий запуск VPP та перевірити наявність інтерфейсів:

vppctl show interface

Скоріш за все ми не побачимо зовнішніх інтерфейсів

```
[root@localhost ~]# vppctl show interface
              Name               Idx    State  MTU (L3/IP4/IP6/MPLS)
          Count
local0                            0     down          0/0/0/0
```

Тому відповідно треба налаштувати їх. Першочергово при запуску VPP він обережно ігнорує інтерфейси, що використовуються ядром Лінукса. Тому перед запуском треба відключити інтерфейси з боку ядра. Наступний крок - задати інтерфейси в конфігураціях VPP. Відповідний пункт dpdk в /etc/vpp/startup.conf

```
dpdk {
        dev 0000:02:01.0
        dev 0000:03:00.0
}
```

Формат заповнення відповідає pci-адресу в системі. Але зауважу, що для правильного розпізнавання потрібно прив’язати драйвери, що підтримуються VPP. Тому в моєму випадку треба позбутися нав’язаного драйверу e1000. Для цього в налаштуваннях віртуальних машин (.vmx файли) змінюємо значення параметру ethernet1.virtualDev на “vmxnet3”, та після перезапуску віртуальної машини перев’язуємо драйвер:

driverctl -v set-override 0000:03:00.0 uio_pci_generic

Можна виконати старт VPP та переконатися, що система підхопила інтерфейс:

```
[root@localhost ~]# vppctl show interface
              Name               Idx    State  MTU (L3/IP4/IP6/MPLS)
          Count
GigabitEthernet3/0/0              1     down         9000/0/0/0
local0                            0     down          0/0/0/0
```

Опісля конфігурації мережевих інтерфейсів робимо фінальні зміни в конфігурацію при запуску.

```
unix {
        nodaemon
        log /var/log/vpp/vpp.log
        full-coredump
        cli-listen /run/vpp/cli.sock
        gid vpp
}

api-segment {
        gid vpp
}

dpdk {
        dev 0000:02:01.0
        dev 0000:03:00.0
}

 plugins {
       ## Disable all plugins, selectively enable specific plugins
       ## YMMV, you may wish to enable other plugins (acl, etc.)
       plugin default { disable }
       plugin dpdk_plugin.so { enable }
       plugin nat_plugin.so { enable }
 }
```

Тому можна з впевненістю запускати VPP та виконати налаштування NAT плагіну

set int state GigabitEthernetb/0/0 up

set dhcp client intfc GigabitEthernetb/0/0 hostname vppgate

loop create

set int l2 bridge loop0 1 bvi

set int ip address loop0 192.168.99.1/24

set int state loop0 up

set int l2 bridge GigabitEthernet3/0/0 1

set int state GigabitEthernet3/0/0 up

create tap host-if-name lstack host-ip4-addr 192.168.99.2/24 host-ip4-gw 192.168.99.1

set int l2 bridge tap0 1

set int state tap0 up

nat44 plugin enable

nat44 add interface address GigabitEthernetb/0/0

set interface nat44 in loop0 out GigabitEthernetb/0/0

Та переконуватися в працездатності NAT серверу на прикладі команди пінгу до серверів поза нашою внутрішньою мережею.

```
12: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
    link/ether 00:0c:29:15:d4:af brd ff:ff:ff:ff:ff:ff
    inet 192.168.99.10/24 brd 192.168.99.255 scope global nopref
       valid_lft 43060sec preferred_lft 43060sec
    inet6 fe80::a296:212c:9e16:58d/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@localhost v2.97]# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=15.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=15.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=15.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=15.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=116 time=15.2 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 15.098/15.283/15.544/0.145 ms
```

Налаштування для ipfw будуть виглядати так:

*ipfw nat 1 config if em0 redirect tcp 192.168.99.2/24*

*ipfw add 100 nat 1 ip4 from any to me in via em0*

*ipfw add 200 nat 1 ip4 from 192.168.31.0/24 to any out via em0*

*ipfw add allow ip from any to any*

Ланцюжок правил для iptables:

*iptables -t nat -A POSTROUTING -o ens32 -j MASQUERADE*

*iptables -A FORWARD -i ens32 -o ens33 -m state --state RELATED,ESTABLISHED -j ACCEPT*

*iptables -A FORWARD -i ens33 -o ens32 -j ACCEPT*

### 3.3. Сценарій та конфігурація TRex

Оскільки звичайний пінг не є сильно навантажувальним засобом перевірки на продуктивність роботи шлюзу, виникає необхідність використання спеціалізованих пакетів програмного забезпечення. В нашому випадку найкращим варіантом буде генератор трафіку. Поміж доступних і безкоштовних засобів я обрав TRex компанії Cisco.

Коротко оглядаючи причини по яким було обрано його, основними можна визначити:

1. Повна емуляція трафіку L3-7 трафіку;
2. Можливість налаштування потоку; генерації пакетів основних мережевих протоколів (ICMP, DNS, HTTP і тд.);
3. Масштабованість потоку
4. Робота на основі бібліотек DPDK;
5. Мультиплатформеність;

Можливості пакету дозволяють мені згенерувати трафік, що буде емулювати обмін даними між заданою кількістю клієнтів у внутрішній мережі та заданою кількістю серверів з зовнішньої мережі. Таким чином я маю можливість регулювати кількість сесій, які створює один користувач.

Також маю зазначити зручну систему конфігурування потоку завдяки файлам формату YAML, через які можна зібрати свій потік з вже готових pcap-файлів, що містять шаблони специфічних пакетів.

Сценарій потоку трафіку буде відповідати профілю IMIX. Основна його відмінність це суміщення в основному потоці трьох субпотоків відповідно:

1. 60-байтовий UDP потік;
2. 590-байтовий UDP потік;
3. 1514-байтовий UDP потік;

Це дозволить зробити більш-менш реалістичний потік з яким стикнеться наш шлюз в реальних умовах.

Оскільки трафік, що генерується, має не відповідати реальним мережам інтерфейсів, то TRex генерує трафік під виглядом того, що він від клієнтів мережі 16.0.0.0/24 та надсилає його до серверів мережі 42.0.0.0/24. Відповідно VPP проводить NAT транслювання та маршрутизацію до другого інтерфейсу серверу TRex.

Конфігурація інтерфейсів TRex:

*- version: 2*

*interfaces: ['03:00.0', '0b:00.0']*

*port_info:*

*- ip: 192.168.99.10*

*default_gw: 192.168.99.1 ### Інтерфейс, що генерує трафік*

*- ip: 192.168.31.201*

*default_gw: 192.168.31.1 ### Інтерфейс, що отримує та аналізує трафік*

Фрагмент конфігурації генерації трафіку:

*distribution : "seq"*

*clients_start : "16.0.0.10"*

*clients_end : "16.0.0.110"*

*servers_start : "48.0.0.1"*

*servers_end : "48.0.0.100"*

Отже ми згенеруємо трафік в ході якого 100 користувачів створять по 100 сесій до різних серверів.

Особливістю в швидкій обробці трафіку, що надходить до TRex, є не повна перевірка пакету, а звірк вихідним пакетом. Це дозволяє не витрачати обчислювальні ресурси на декапсуляціюа його чексум з пакетів, та витрачати на генерування потоку.

Трафік що йде до шлюзу є TX трафіком, а з шлюзу - RX трафіком. Різниця між ними буде вираховуватися як Drop rate. Затримки же між відправленням і отриманням пакету називаються window, з них вже йде вимір latency.

Для операційної системи для TRex серверу я обрав все той же Centos 7. Оскільки він підтримує DPDK бібліотеки і тому проблем з використанням пакету TRex на ньому не буде.

### 3.4. Критерії оцінки продуктивності

В даному випадку під продуктивністю ми маємо на увазі цілий ряд критеріїв, що позначають швидкодію та ефективність роботи досліджуваного інтернет-шлюзу на основі NAT протоколу.

**Відношення швидкості вхідного та вихідного трафіку** - за допомогою порівняння швидкостей входу та виходу пакетів із мережевого шлюзу, ми оцінюємо пропускну здатність шлюзу в конкретних умовах навантаження вхідним трафіком. Наближення значення до 1 цього критерію збільшує продуктивність системи.

**Швидкість втрати пакетів** - втрачений пакет, це пакет що не був оброблений, а тому з швидкості втрати випливає частка пакетів, яку мережевий шлюз може транслювати поки останній чекає на обробку. Відповідно зменшення цього критерію збільшить продуктивність системи.

**Середня та максимальна затримка** - часовий проміжок між двома трансльованими пакетами. Цей критерій дає якісно зрозуміти скільки часу займає трансляція пакету. Максимальна затримка буде відповідати сценарію створення нової сесії. Зниження цих критеріїв є підвищенням продуктивності.

**Обмеження в кількості сесій на клієнта** - ресурси системи не є безлімітними, а величина комбінацій між серверами, до яких може надіслати пакет користувач, і вільними портами користувача зазвичай виходить за межі доступних портів зовнішньої адреси провайдера, що обслуговує декілька користувачів. Збільшення цього критерію є підвищенням продуктивності. Але ы прямий контроль цього параметру є адміністративною можливістю.

**Рівномірність та динамічність розподілу сесій по ір-адресам** - маючи в наявності більше ніж одну зовнішню адресу, виникає потреба в рівномірному розподілі сесій між ними. Статичний розподіл призведе до повного використання портового простору ір-адреси, в той час як сусідня матиме навпаки майже вільний простір. Подальший розподіл вимагатиме додаткових обчислень, а також збільшує можливість помилок при створенні нових сесій. Числова оцінка цього критерію полягає у максимальній різниці

між використаним портовим простором між довільними зовнішніми адресами.

### 3.5. Отриманні результати та їх аналіз

Типічний результат опісля проведення тестування матиме вигляд такої таблиці

```
port : 0
------------
 opackets                                : 101033
 obytes                                  : 6674614
 ipackets                                : 54
 ibytes                                  : 3672
 Tx :     351.50 Kbps
port : 1
------------
 opackets                                : 100864
 obytes                                  : 7359176
 ipackets                                : 8607
 ibytes                                  : 607144
 Tx :     350.50 Kbps
 Cpu Utilization : 6.1  %  0.0 Gb/core
 Platform_factor : 1.0
 Total-Tx        :     702.00 Kbps   NAT time out    :      255
 Total-Rx        :       1.08 Kbps   NAT aged flow id:        0
 Total-PPS       :       1.33 Kpps   Total NAT active:        0
 Total-CPS       :       1.73  cps   Total NAT opened:      277

 Expected-PPS    :     102.71  pps
 Expected-CPS    :       2.78  cps
 Expected-BPS    :     767.80 Kbps

 Active-flows    :        0  Clients :       87   Socket-util : 0.0000 %
 Open-flows      :      277  Servers :       10   Socket :        0 Socket/Clients :  0.0
 drop-rate       :     700.92 Kbps
 summary stats
 --------------
 Total-pkt-drop       : 193236 pkts
 Total-tx-bytes       : 14033790 bytes
 Total-tx-sw-bytes    : 13259532 bytes
 Total-rx-bytes       : 610816 byte

 Total-tx-pkt         : 201897 pkts
 Total-rx-pkt         : 8661 pkts
 Total-sw-tx-pkt      : 200902 pkts
 Total-sw-err         : 0 pkts
 Total ARP sent       : 4 pkts
 Total ARP received   : 2 pkts
 maximum-latency   : 9677 usec
 average-latency   : 142 usec
 latency-any-error : ERROR
```

Поля що для нас мають найбільше значення це:

1. opackets - кількість пакетів що покинули порт;
2. ipackets - кількість пакетів що надійшли до порту;

3. Total-Tx - швидкість відправлення пакетів;

4. Total-Rx - швидкість отримання пакетів;

5. drop-rate - швидкість втрати пакетів (дропів);

6. maximum-latency - максимальна затримка;

7. average-latency - середня затримка;

Провівши 5 тестів протягом 15 хвилин для кожного варіанту реалізації мережевого шлюзу та вирахувавши середні значення маємо:

| | VPP | Ipfw nat | Iptables |
|---|---|---|---|
| opackets | 903643 | 906818 | 907739 |
| ipackets | 562017 | 419823 | 70757 |
| Total-Tx | 332.36 Kbps | 381.39 Kbps | 689.41 Kbps |
| Total-Rx | 208.35 Kbps | 179.32 Kbps | 635.27 bps |
| drop-rate | 124.01 Kbps | 202,07 Kbps | 688.77 Kbps |
| maximum-latency | 97 usec | 74 usec | 275410 usec |
| average-latency | 45 usec | 45 usec | 141 usec |

Отже можна судити по кожному із критеріїв, що стосуються швидкості та якості обробки трафіку.

Відношення швидкості вхідного та вихідного трафіку для VPP буде складати 1.6, в той час як для ipfw - 2.13 . Це підкреслює зростання пропускної здатності на 16% у VPP.

Також маємо яскраву перевагу VPP в зменшенні швидкості втрати пакетів на 37%.

При вище зазначених числових перевагах VPP ми бачимо більше значення максимальної затримки. Я вважаю, що це затримка між двома векторами пакетів, оскільки відбувається збирання наступного вектора, що вимагає певного часу. Але як бачимо в середньому це не змінило затримку.

Iptables показав більш ніж 10-кратно менші показники пропускної здатності. При детальному розгляді формування результатів в реальному часі, я зробив висновок, що в перший етап відкривається в середньому 25 сесій та в подальшому трансляція не відбувається. Допущу таке пояснення: при інтенсивному навантаженні відбувається помилка в буфері мапінгу, що не дає створити нові сесії.

Для оцінки розподілу сесій між адресними просторами, треба використовувати внутрішню статистику, що надають реалізації інтернет шлюзи. І тут варто зазначити, що вивід інформації про сесії в ipfw є малоінформативним в порівнянні з VPP. Наприклад:

*ipfw nat show log*

*nat 1: icmp=4, udp=298, tcp=0, sctp=0, pptp=0, proto=0, frag_id=0 frag_ptr=0 / tot=302*

В той час як VPP надає набагато більше інформації

```
vpp# show nat44 summary
max translations per thread: 64512 fib 0
transitory tcp LRU min session timeout 10801 (now 14166)
udp LRU min session timeout 1093 (now 14166)
unknown protocol LRU min session timeout 9613 (now 14166)
total sessions: 51 (timed out: 48)
tcp sessions:
    total: 35 (timed out: 33)
        established: 0 (timed out: 0)
        transitory: 35 (timed out: 33)
udp sessions:
    total: 11 (timed out: 11)
icmp sessions:
    total: 0 (timed out: 0)
other sessions:
    total: 5 (timed out: 4)
```

Та конкретно по сесії:

```
    i2o 192.168.99.2 proto TCP port 60354 fib 0
    o2i 192.168.31.44 proto TCP port 60354 fib 0
       external host 35.224.170.84:80
       i2o flow: match: saddr 192.168.99.2 sport 60354 daddr 35.224.170.84 dport 80 proto TCP fib_idx 0
rewrite: saddr 192.168.31.44 sport 60354 daddr 35.224.170.84 dport 80 txfib 0
       o2i flow: match: saddr 35.224.170.84 sport 80 daddr 192.168.31.44 dport 60354 proto TCP fib_idx 0
 rewrite: daddr 192.168.99.2 dport 60354 txfib 0
       index 2
       last heard 12728.02
       timeout in -1033.48
       total pkts 11, total bytes 823
       dynamic translation
```

Повертаючись до аналізу, впевнено видно, що VPP виконує розподілення відповідно до вказаного алгоритму. В той час як важко прослідкувати розподілення в даній конфігурації ipfw, зробивши тестування з різною кількістю користувачів та виділених ip-адрес, можна прослідкувати статичне розподілення.

Отже тестування VPP та ipfw показало перевагу першого в продуктивності і ефективного використання ресурсів.

# ВИСНОВОК

В процесі виконання кваліфікаційної роботи було розглянуто мережевий шлюз гуртожиткового інтернету та поточні проблеми з його конфігурацією. На основі цього я дослідив варіанти можливих рішень та виділив перспективне у вигляді розгортання інтернет-шлюзу засобами пакету VPP.

Запропоноване мною архітектурне рішення має підвищити продуктивність мережевого шлюза в процесі транслювання внутрішніх IP-адрес та дозволить реалізувати функціонал для подальшого розвитку мережі.

Для підтвердження моїх міркувань був зібраний тестовий сценарій домашнього маршрутизатора. В ньому було протестовано конфігурації на основі поточного та запропонованого програмного забезпечення. Використання пакету VPP дозволяє отримати приріст пропускної здатності до 16% у порівнянні зі старою конфігурацією. В ряді інших параметрів VPP має рівні або більші показники.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is VPP? [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.fd.io/view/VPP/What_is_VPP%3F

2. Tech Preview: CNF Acceleration Evaluation Using Nvidia Accelerated Network Computing. - NVIDIA Networking Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nvidia.com/networking/pages/releaseview.action?pageId=39267694

3. VPP/NAT [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.fd.io/view/VPP/NAT

4. Olaf Kirch, Terry Dawson Linux Network Administrator's Guide, Second Edition : навч. посіб. Sebastopol : O'Reilly Media, Inc., 2000. 442 с

5. Paco Hope, Bruce Potter, Yanek Korff Mastering FreeBSD and OpenBSD Security : навч. посіб. Sebastopol : O'Reilly Media, Inc., 2005. 407 с

6. Anthony Sabella, Rik Irons-Mclean Orchestrating and Automating Security for the Internet of Things: Delivering Advanced Security Capabilities from Edge to Cloud for IoT : навч. посіб. Indianapolis : Cisco Press, 2018. 322 с