УДК 621.391
DOI https://doi.org/10.32782/2663-5941/2026.1.1/16

Приймак С.О.
https://orcid.org/0009-0008-7532-6095
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Кравчук С.О.
https://orcid.org/0000-0002-4118-0226
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ПРАКТИЧНА РЕАЛІЗАЦІЯ ВИСОКОДОСТУПНОЇ VOIP СИСТЕМИ.

У статті розглянуто практичну реалізацію VoIP-систем (Voice over IP) підвищеної доступності, яка забезпечують голосовий зв'язок через IP-мережі. Наведено приклад практичної реалізації повної VoIP-системи, зокрема налаштування та тестування основних компонентів для забезпечення безперервності сервісу, відмовостійкості, масштабованості та ефективності використання ресурсів.

Запропонований метод практичної реалізації VoIP-системи базується на використанні безкоштовного програмного забезпечення Kamailio (сервер сигналізації SIP), RTPengine (обробка медіа-трафіку) та Redis (швидкісна база даних для зберігання RTP-транзакцій). Всі компоненти базуються на платформі Linux. Реалізована архітектура включає механізми автоматичного резервування серверів сигналізації та медіа-шлюзів за допомогою протоколу VRRP, балансування навантаги та безшовного перемикання на резервні вузли у разі відмови одного або декількох компонентів.

Це суттєво відрізняється від традиційних підходів, де відмова одного з ключових елементів (сервер сигналізації або медіа-шлюз) призводить до негайного завершення всіх активних з'єднань.

Виконано експериментальне моделювання роботи системи, яке підтвердило її високу ефективність: навіть у разі відмови 50% компонентів система зберігає працездатність і не переривает активні голосові виклики.

Отримані результати можуть бути корисними для розробників і операторів VoIP-мереж, операторів стільникового зв'язку, а також підприємств, які використовують IP-телефонію для критично важливих комунікацій. Запропонований підхід до реалізації системи дозволяє мінімізувати ризики переривання зв'язку, підвищити якість обслуговування користувачів і забезпечити високу надійність роботи VoIP-сервісів.

Ключові слова: *VoIP система, SIP, Kamailio, RTP, VRRP, висока доступність, надійність.*

Постановка проблеми. Практика експлуатації VoIP-інфраструктур свідчить, що навіть короткочасна відмова сигнального або медіа-вузла може призвести до:

– повної втрати сервісу;

– розриву активних голосових з'єднань;

– неможливості реєстрації абонентських пристроїв;

– зниження якості обслуговування (QoS) та довіри користувачів.

Наукова проблема, що розглядається у статті, формулюється наступним чином:

необхідно розробити та практично реалізувати архітектуру високодоступної VoIP-системи, яка забезпечує безперервну роботу SIP-сигналізації та RTP-медіапотоків у разі відмови одного або декількох вузлів системи, із мінімальним часом перемикання та без втрати функціональності.

Для розв'язання зазначеної проблеми необхідно вирішити такі завдання:

– організувати резервування сигнальних серверів SIP;

– забезпечити синхронізацію реєстрацій та сигнального стану між вузлами;

– реалізувати відмовостійку обробку RTP-трафіку;

– мінімізувати вплив відмов на активні голосові з'єднання;

– використати відкриті програмні засоби з можливістю масштабування.

У зв'язку з цим актуальним є завдання розробки та впровадження високодоступних (High Availability, HA) VoIP-систем, здатних забезпечувати безперервну роботу сервісу навіть у разі відмови окремих компонентів.

Об'єктом дослідження у даній роботі є процеси забезпечення відмовостійкості та високої доступності VoIP-систем у IP-мережах.

Предметом дослідження є методи, механізми та архітектурні рішення побудови високодоступної VoIP-системи з використанням відкритого програмного забезпечення. Традиційні VoIP-рішення базуються на централізованій архітектурі, у якій сигнальний сервер (SIP-сервер) та медіашлюз є єдиними точками обробки трафіку. Такий підхід характеризується наявністю єдиної точки відмови (Single Point of Failure), що є неприйнятним для систем, які надають критично важливі послуги.

Існуючі комерційні рішення високодоступних VoIP-систем є дорогими у впровадженні, закритими для модифікації та складними в інтеграції з іншими мережевими сервісами.

У той же час, відкриті програмні платформи, такі як Kamailio [1], RTPengine [2], Redis [3], Keepalived [4], надають широкі можливості для побудови гнучких та масштабованих систем, проте їх комплексне застосування потребує ретельного опрацювання та узгодження механізмів взаємодії.

Таким чином, існує науково-практична проблема, яка полягає у відсутності типових, перевірених на практиці архітектур високодоступних VoIP-систем, що базуються на відкритому програмному забезпеченні та забезпечують автоматичне перемикання при відмовах, збереження сигнального стану та безперервність передачі медіапотоків.

Постановка завдання. Метою дослідження є підвищення надійності та доступності VoIP-систем шляхом розробки і впровадження високодоступної архітектури сигналізації та медіаобробки.

Дана робота є продовженням досліджень, які вже були опубліковані в [5].

Аналіз останніх досліджень і публікацій. В [9] розглянуто приклад побудування VoIP-системи з шифруванням голосового та сигнального трафіку в режимі реальному часі,

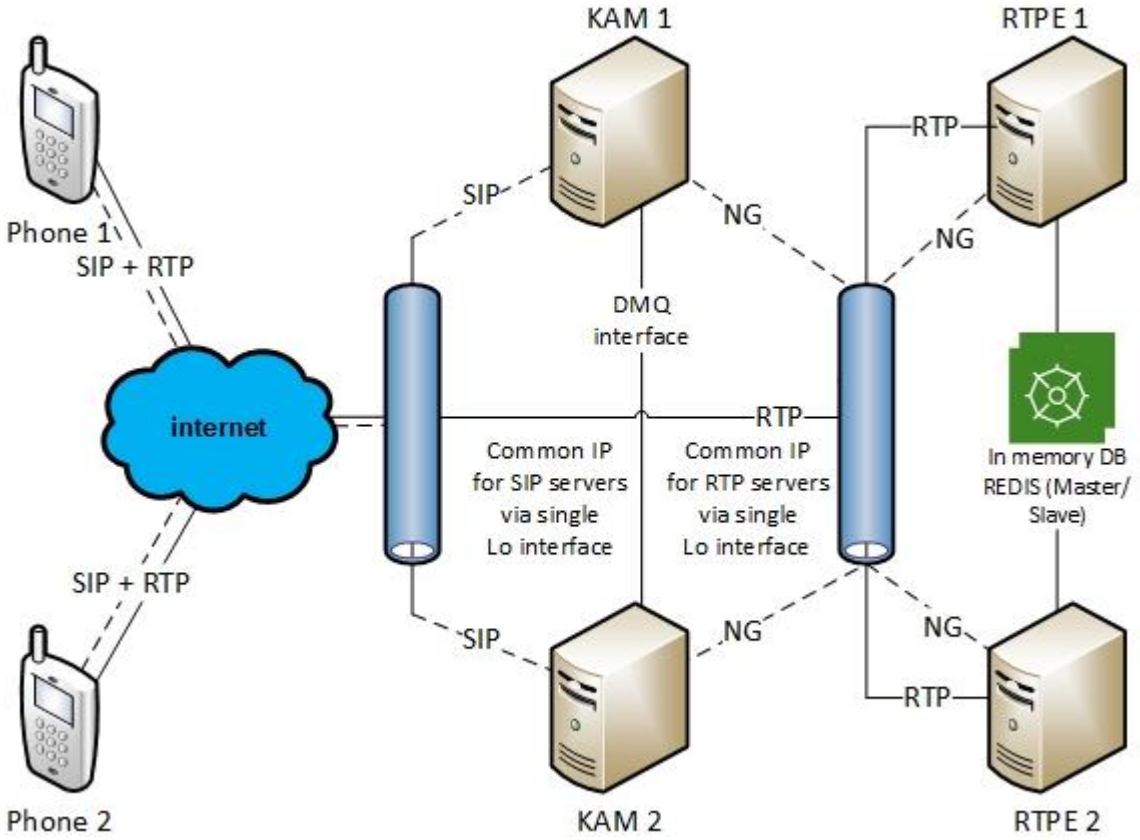


Рис.1. Схема високодоступної VoIP системи

в [10] приділено увагу побудуванню безпечної комп'ютерної мережі для VoIP-системи та наведені типові показники навантаги серверів VoIP, але в жодній з цих робот не розглянуто аспекти стійкості до збою програмних та апаратних елементів VoIP-системи. В [11] розглянуто декілька методів підвищення доступності VoIP-системи, але жоден з них не виконує резервування існуючих голосових потоків RTP. У випадку реалізації запропонованої нами системи вплив на сервіси відсутній навіть при відмові 50 % її компонентів. Представляє інтерес для поліпшення VoIP-систем підходи на базі безкоштовного програмного забезпечення Kamailio, RTPengine та Redis, але у відомих роботах їх застосування недостатньо вивчене.

Завдання:

– Розробити архітектуру кластерної SIP-сигналізації на основі Kamailio.

– Реалізувати механізм синхронізації реєстрацій та стану викликів.

– Побудувати відмовостійкий кластер RTP-медіашлюзів.

– Забезпечити автоматичне перемикання між вузлами при відмовах.

– Провести практичну перевірку працездатності запропонованого рішення.

Виклад основного матеріалу дослідження. Теоретичні положення були викладені в [5] та [6]. Схема вискодоступної VoIP системи наведена на рис. 1.

1. Налаштування серверів сигналізації KAM1 та KAM2.

1.а. Налаштовуємо мережевий інтерфейс.

Для цього, в файл /etc/netplan/net.yaml вносимо наступні дані:

```
network:
    version: 2
    renderer: networkd
    ethernets:
        enp0s3:
            dhcp4: false
Для KAM1:  addresses: [10.0.2.4/24]        Для KAM2:  addresses: [10.0.2.5/24]
            nameservers:
                addresses:
                - 8.8.8.8
                - 8.8.4.4
                search: []
            optional: true
            routes:
              - to: default
                via: 10.0.2.1
                metric: 100

              - to: 10.20.20.2
                via: 10.0.2.10
                metric: 100
    dummy-devices:
        lo0:
          addresses:
            - 10.20.20.1/32
```

Тут в лістингу:

10.0.2.4 – IP мережевого інтерфейса KAM1;

10.0.2.5 – IP мережевого інтерфейса KAM2;

- 8.8.8.8 та 8.8.4.4 – DNS сервера;
- 10.20.20.1 – адрес loopback інтерфейсів KAM1 та KAM2;
- 10.0.2.1 – IP шлюз;
- 10.0.2.10 – "плаваючий" IP медіа шлюзів RTPE1 та RTPE2;
- 10.20.20.2 - адрес loopback інтерфейсів RTPE1 та RTPE2.

Для активації змін в конфігурації виконуємо команду:

```
# netplan apply
```

Результатом виводу повинна бути відсутність помилок.

1.б. Встановлюємо програмне забезпечення (ПЗ) для контролю працездатності вузлів КАМ1 та КАМ2:

```
# apt install sipsak
```

```
root@ip-172-31-82-47:/home/ubuntu# apt install sipsak
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcares2
The following NEW packages will be installed:
  libcares2 sipsak
0 upgraded, 2 newly installed, 0 to remove and 18 not upgraded.
Need to get 125 kB of archives.
After this operation, 309 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libcares2 amd64 1.27.0-1.0ubuntu1 [73.7 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 sipsak amd64 0.9.8.1-1build4 [50.9 kB]
Fetched 125 kB in 0s (1314 kB/s)
Selecting previously unselected package libcares2:amd64.
(Reading database ... 156180 files and directories currently installed.)
```

```
# apt install keepalived
```

```
root@ip-172-31-87-8:/home/ubuntu# apt install keepalived
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ipvsadm libsnmp-base libsnmp40t64
Suggested packages:
  heartbeat ldirectord snmp-mibs-downloader
The following NEW packages will be installed:
  ipvsadm keepalived libsnmp-base libsnmp40t64
0 upgraded, 4 newly installed, 0 to remove and 29 not upgraded.
Need to get 1772 kB of archives.
After this operation, 5941 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 libsnmp-base all 5.9.4+dfsg-1.1ubuntu3.1 [206 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 libsnmp40t64 amd64 5.9.4+dfsg-1.1ubuntu3.1 [1066 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 keepalived amd64 1:2.2.8-1build2 [460 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 ipvsadm amd64 1:1.31-1ubuntu0.1 [40.3 kB]
Fetched 1772 kB in 0s (14.3 MB/s)
Selecting previously unselected package libsnmp-base.
(Reading database ... 130443 files and directories currently installed.)
```

Та виконуємо їх налаштування. В файл /etc/keepalived/keepalived.conf вносимо наступні дані для КАМ1 та КАМ2:

```
vrrp_script check_sip {
    script       "/etc/keepalived/checksip.sh"
    interval 1   # check every 2 seconds
    fall 2       # requires 2 failures for KO
    rise 2       # requires 2 successes for OK
    weight -60
}
bfd_instance kamalio {
```

Для КАМ1:

```
    neighbor_ip  10.0.2.5  # remote_node_ip
    source_ip    10.0.2.4  # local_ip
```

Для КАМ2:

```
    neighbor_ip  10.0.2.4  # remote_node_ip
    source_ip    10.0.2.5  # local_ip
}
```

```
vrrp_instance KAMAILIO {
```

Для KAM1:

```
 state MASTER
 priority 150
```

Для KAM1:

```
 state BACKUP
 priority 100

 interface enp0s3
 virtual_router_id 56
 advert_int 1
 authentication {
     auth_type PASS
     auth_pass somepassword
 }
 virtual_ipaddress {
     10.0.2.9/24 dev enp0s3
 }

 track_bfd {
      kamalio weight 60 reverse
 }
 track_script {
   check_sip
 }
}
```

Де в лістингу:
- 10.0.2.5 - IP мережевого інтерфейса KAM2
- 10.0.2.9 – “плаваючий” IP серверів сигналізації KAM1 та KAM2.

Створюємо командний файл та робимо його виконавчим:

```
# touch /etc/keepalived/checksip.sh
# chmod +x /etc/keepalived/checksip.sh
```

Результатом виводу повинна бути відсутність помилок.
Та вносимо в нього наступні дані:

```
#!/bin/bash

node01=10.0.2.4
node02=10.0.2.5
```

Для KAM1:

```
return_code=0 # success
```

Для KAM2:

```
return_code=1 # fail

timeout 1 sipsak -s sip:$node01:5060
exit_status=$?
if [[ $exit_status -eq 0 ]]; then
  echo "sip ping successful to node01 [$node01]"
  exit $return_code
fi
```

```
timeout 2 sipsak -s sip:$node02:5060
exit_status=$?
if [[ $exit_status -eq 0 ]]; then
  echo "sip ping successful to node02 [$node02]"

   Для KAM1:
  return_code=1

   Для KAM2:
  return_code=0

fi

echo "return code [$return_code]"
exit $return_code
```

Далі виконуємо перезапуск утіліти keepalived з метою використання нею нових налаштувань:

```
# systemctl restart keepalived
```

Та перевіряємо статус:

```
# systemctl status keepalived
```

```
root@ip-172-31-87-8:/home/ubuntu# systemctl status keepalived
o keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/usr/lib/systemd/system/keepalived.service; enabled; preset: enabled)
     Active: inactive (dead)
  Condition: start active            at Mon 2025-05-12 08:17:57 UTC; 3min 18s ago
            └─ ConditionFileNotEmpty=/etc/keepalived/keepalived.conf was not met
       Docs: man:keepalived(8)
             man:keepalived.conf(5)
             man:genhash(1)
             https://keepalived.org
```

1.в. Виконуємо інсталяцію сервера сигналізації Kamailio (KAM1 та KAM2).

Завантажуємо необхідне ПЗ для компіляції:

```
# apt install git-core  gcc g++ flex bison make libssl-dev lua5.4 liblua5.4-dev libhiredis-dev
```

```
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-1024-aws
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-1027-aws.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart networkd-dispatcher.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #1: sshd[1031]
 ubuntu @ user manager service: systemd[1378]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

Інсталюємо SIP сервер Kanailio. Для цього завантажуємо код програми з офіційного сайту та розпаковуємо файли з архіву:

```
# wget http://www.kamailio.org/pub/kamailio/latest-6.0.x/src/kamailio-6.0.1_src.tar.gz
```

```
root@ip-172-31-87-8:/home/ubuntu# wget http://www.kamailio.org/pub/kamailio/latest-6.0.x/src/kamailio-6.0.1_src.tar.gz
--2025-05-12 08:28:27--  http://www.kamailio.org/pub/kamailio/latest-6.0.x/src/kamailio-6.0.1_src.tar.gz
Resolving www.kamailio.org (www.kamailio.org)... 193.22.119.66, 2a00:d60:0:400::2
Connecting to www.kamailio.org (www.kamailio.org)|193.22.119.66|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13064707 (12M) [application/x-gzip]
Saving to: 'kamailio-6.0.1_src.tar.gz'

kamailio-6.0.1_src.tar.gz                          100%[=================================================

2025-05-12 08:29:01 (379 KB/s) - 'kamailio-6.0.1_src.tar.gz' saved [13064707/13064707]

root@ip-172-31-87-8:/home/ubuntu#
```

```
# tar xvfz kamailio-6.0.1_src.tar.gz && cd kamailio
```

```
kamailio-6.0.1/src/modules/rtimer/doc/
kamailio-6.0.1/src/modules/rtimer/doc/rtimer_admin.xml
kamailio-6.0.1/src/modules/rtimer/doc/rtimer.xml
kamailio-6.0.1/src/modules/rtimer/doc/Makefile
kamailio-6.0.1/src/modules/rtimer/README
kamailio-6.0.1/src/modules/rtimer/rtimer_mod.c
kamailio-6.0.1/src/modules/rtimer/CMakeLists.txt
kamailio-6.0.1/src/modules/rtimer/Makefile
kamailio-6.0.1/src/Makefile.utils
kamailio-6.0.1/src/Makefile.vars
kamailio-6.0.1/src/Makefile
root@ip-172-31-87-8:/home/ubuntu#
```

Створюємо конфігураційний файл

```
# cd kamailio-6.0.1 && make cfg
```

```
root@ip-172-31-87-8:/home/ubuntu/kamailio-6.0.1# make cfg
make -C src/ cfg
make[1]: Entering directory '/home/ubuntu/kamailio-6.0.1/src'
target architecture <x86_64>, host architecture <x86_64>
making config...
rm -f modules.lst
make --no-print-directory modules.lst
saving modules list...
make[1]: Leaving directory '/home/ubuntu/kamailio-6.0.1/src'
```

Активуємо необхідні внутрішні модулі Kamailio.

Для цього редагуємо строку include_modules в файлі src/modules.lst:

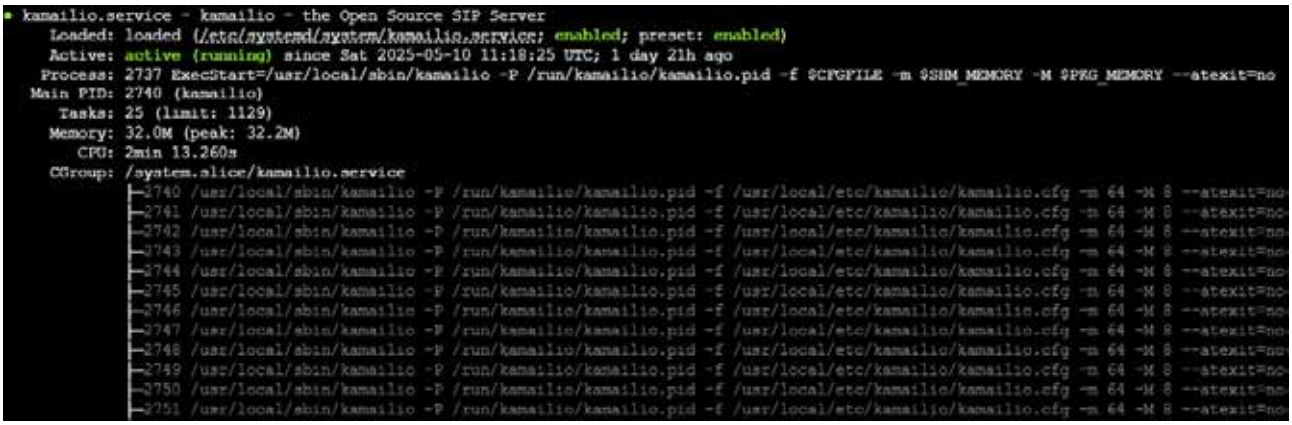
```
include_modules = dmq ndb_redis app_lua
```

Виконуємо компіляцію програми:

```
# make all && make install
```

Виконуємо запуск Kamailio та перевіряємо статус:

```
# systemctl start kamailio
# systemctl status kamailio
```



1.г. Виконуємо налаштування серверів сигналізації KAM1 та KAM2.

Для цього редагуємо конфігураційний файл /usr/local/etc/kamailio/kamailio.cfg та налаштовуємо IP адреси, на яких буде проходить обмін сігнальними повідомленнями SIP:

Для KAM1:

```
listen=udp:10.0.2.4:5060
listen=udp:10.0.2.4:5062
```

Для KAM2:

```
listen=udp:10.0.2.5:5060
listen=udp:10.0.2.5:5062
```

Для KAM1 та KAM2:

```
listen=udp:10.20.20.1:5060
advertised_address="10.20.20.1"
```

В секції loadmodule підгружаємо додаткові внутрішні модулі:

```
loadmodule "dmq.so"
loadmodule "dmq_usrloc.so"
loadmodule "htable.so"
loadmodule "rtpengine.so"
loadmodule "sdpops.so"
loadmodule "auth.so"
loadmodule "dmq_usrloc.so"
loadmodule "ipops.so"
loadmodule "ndb_redis.so"
```

Та в секції modparam виконуємо їх налаштування.

Реплікація сигнальних повідомлень між серверами

Для KAM1:

```
modparam("dmq", "server_address", "sip:10.0.2.:5062")
modparam("dmq", "notification_address", "sip:10.0.2.5:5062")
```

Для KAM2:

```
modparam("dmq", "server_address", "sip:10.0.2.5:5062")
modparam("dmq", "notification_address", "sip:10.0.2.4:5062")
```

Для KAM1 та KAM2:

```
modparam("dmq", "multi_notify", 1)
modparam("dmq", "num_workers", 4)
modparam("dmq", "ping_interval", 15)
modparam("dmq_usrloc", "enable", 1)
modparam("htable", "enable_dmq", 1)
modparam("htable", "dmq_init_sync", 1)
```

Активація модуля обробки голосового трафіку RTP

```
modparam("rtpengine", "rtpengine_sock", "udp:10.20.20.2:2223")
```

Налаштування серверів KAM1 та KAM2 на роботу з БД Redis для маршрутизації голосових викликів і реєстрації користувачів:

```
modparam("ndb_redis", "server", "name=MyRedisServer;addr=10.0.2.15;port=6379;db=3")
modparam("ndb_redis", "server", "name=RedisUser;addr=10.0.2.15;port=6379;db=4")
modparam("ndb_redis", "init_without_redis", 1)
```

Завдамо логіку опрацювання аутентентифікації користувача та реєстрації в системі з наступною реплікацією даних на резервний сервер. Для цього в розділ request_route додамо наступне:

```
if (method == "REGISTER") {
 redis_cmd("RedisUser", "HMGET $fU password", "r");
    if (!pv_www_authenticate("$fd", "$redis(r=>value[0])", "0")) {
          auth_challenge("$fd", "1");
          exit;
    }
    route(REGISTRAR);
    dmq_t_replicate();
    exit;
}
if ($rm == "KDMQ" && $rp == 5062) {
      dmq_handle_message();
      exit;
}
```

Додаємо блок маршрутизації голосових викликів:

```
 if (is_method("INVITE|SUBSCRIBE")) {
        if (has_body("application/sdp")) {
          sdp_keep_codecs_by_name("PCMA,PCMU,telephone-event");
     if(!msg_apply_changes()){
       xlog("Error changing codecs......");
     }
     if(!lookup("location")) {
       $var(j) = $(rU{s.len});
             $var(s) = "";
             $avp(dst_IP) = "";
             while($var(j) >= 3) {
               $var(s) = $(rU{s.substr, 0, $var(j)});
               redis_cmd("MyRedisServer", "GET $var(s)", "r");
               $avp(dst_IP) = $redis(r=>value);
               if( is_ip($avp(dst_IP)) && $avp(dst_IP) != "") {
                     $rd = $avp(dst_IP);
                     xlog("Value of Dst_IP is:  $avp(dst_IP)");
```

```
                        redis_free("r");
                        break;
                      }
                      $var(j) = $var(j) - 1;
                    }
                    if($avp(dst_IP) == "") {
                      xlog("Dst_IP not found or incorrect!");
            sl_send_reply("404", "Not Found");
                    }
        }
           }
      record_route();
    }
```

Активуємо логіку обробки голосових пакетів RTP. В функції route[RELAY] та onreply_ route[MANAGE_REPLY] додаєм наступний код:

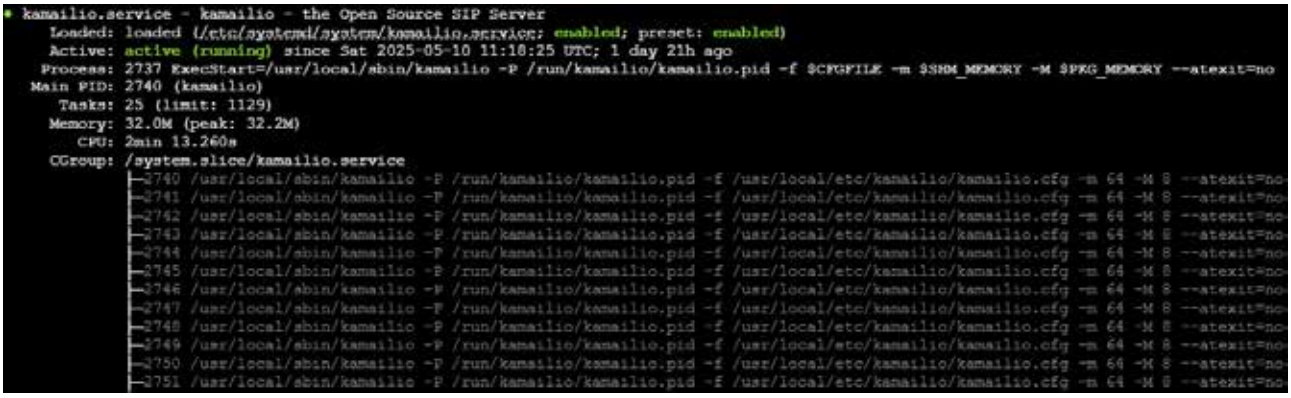
```
  if (has_body("application/sdp")) {
      rtpengine_manage("replace-origin replace-session-connection");
  }
```

1.д. Виконуємо перезапуск ПЗ Kamailio з метою активації вищенаведених налаштувань та перевірку статусу роботи програми.

```
# systemctl restart kamailio
```

Результатом виводу повинна бути відсутність помилок.

```
# systemctl status kamailio
```



На цьому налаштування сигнальних серверів КАМ1 та КАМ2 можно вважати завершеним.

2. Встановлення та налаштування медіа серверів RTPE1 та RTPE2.

2.а. Налаштовуємо мережевий інтерфейс.
Для цього, в файл /etc/netplan/net.yaml вносимо наступні дані:

```
network:
    version: 2
    renderer: networkd
    ethernets:
        enp0s3:
            dhcp4: false
```

Для RTPE1:

```
        addresses: [10.0.2.11/24]
```

Для RTPE2:

```
        addresses: [10.0.2.12/24]

        nameservers:
            addresses:
            - 8.8.8.8
            - 8.8.4.4
            search: []
        optional: true
        routes:
          - to: default
            via: 10.0.2.1
            metric: 100

 dummy-devices:
     lo0:
       addresses:
         - 10.20.20.2/32
```

Де у лістингу:
- 10.0.2.11 – IP мережевого інтерфейса RTPE1;
- 10.0.2.12 – IP мережевого інтерфейса RTPE2;
- 8.8.8.8 та 8.8.4.4 – DNS сервера;
- 10.20.20.2 – адреса loopback інтерфейсів RTPE1 та RTPE2;
- 10.0.2.1 – IP шлюз;
- 10.0.2.10 – “плаваючий” IP медіа шлюзів RTPE1 та RTPE2;

Для активації змін в конфігурації виконуємо команду:

```
# netplan apply
```

Результатом виводу повинна бути відсутність помилок.

2.б. Встановлюємо програмне забезпечення keepalived для контролю працездатності вузлів RTPE1 та RTPE2:

```
# apt install keepalived
```

```
root@ip-172-31-87-8:/home/ubuntu# apt install keepalived
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ipvsadm libsnmp-base libsnmp40t64
Suggested packages:
  heartbeat ldirectord snmp-mibs-downloader
The following NEW packages will be installed:
  ipvsadm keepalived libsnmp-base libsnmp40t64
0 upgraded, 4 newly installed, 0 to remove and 29 not upgraded.
Need to get 1772 kB of archives.
After this operation, 5941 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 libsnmp-base all 5.9.4+dfsg-1.1ubuntu3.1 [206 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 libsnmp40t64 amd64 5.9.4+dfsg-1.1ubuntu3.1 [1066 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 keepalived amd64 1:2.2.8-1build2 [460 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 ipvsadm amd64 1:1.31-1ubuntu0.1 [40.3 kB]
Fetched 1772 kB in 0s (14.3 MB/s)
Selecting previously unselected package libsnmp-base.
(Reading database ... 130443 files and directories currently installed.)
```

Та виконуємо їх налаштування. В файл /etc/keepalived/keepalived.conf вносимо наступні дані для RTPE1:

```
vrrp_script check_rtpe {
    script       "/etc/keepalived/checkrtpe.sh"
    interval 1   # check every 2 seconds
    fall 2       # requires 2 failures for KO
    rise 2       # requires 2 successes for OK
    weight -60
}

#define BFD session
bfd_instance rtpengine {
     neighbor_ip  10.0.2.12  # remote_node_ip
     source_ip    10.0.2.11  # local_ip
}

vrrp_instance RTPENGINE_1 {
    state MASTER
    interface enp0s3
    virtual_router_id 57
    priority 150
    #nopreempt
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass somepassword
    }

    virtual_ipaddress {
        10.0.2.10/24 dev enp0s3
    }

    #add bfd tracking
    track_bfd {
         rtpengine weight 60 reverse
    }

    track_script {
      check_rtpe
    }
}
```

Де у лістингу 10.0.2.10 – “віртуальний” IP серверів сигналізації RTPE1 та RTPE2.

Відповідно файл checkrtpe.sh містить дані:

```
#!/bin/bash

node01=10.0.2.11
node02=10.0.2.12
port=2225
return_code=0 # success

# check local instance
timeout 1 nc -z $node01 $port > /dev/null 2>&1
exit_status=$?
if [[ $exit_status -eq 0 ]]; then
  echo "sip ping successful to node01 [$node01]"
  exit $return_code
fi
```

```
# check remote instance
timeout 2  nc -z $node02 $port > /dev/null 2>&1
exit_status=$?
if [[ $exit_status -eq 0 ]]; then
  echo "sip ping successful to node02 [$node02]"
  return_code=1
fi

echo "return code [$return_code]"

exit $return_code
```

Переходимо до налаштування RTPE2. В файл /etc/keepalived/keepalived.conf вносимо наступні дані:

```
vrrp_script check_rtpe {
    script       "/etc/keepalived/checksip.sh"
    interval 1   # check every 2 seconds
    fall 2       # requires 2 failures for KO
    rise 2       # requires 2 successes for OK
    weight -60
}

#define BFD session
bfd_instance rtpengine {
     neighbor_ip  10.0.2.11  # remote_node_ip
     source_ip    10.0.2.12  # local_ip
}

vrrp_instance RTPENGINE_2 {
    state BACKUP
    interface enp0s3
    virtual_router_id 57
    priority 100
    #nopreempt
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass somepassword
    }

    virtual_ipaddress {
        10.0.2.10/24 dev enp0s3
    }

    #add bfd tracking
    track_bfd {
         rtpengine weight 60 reverse
    }

    track_script {
      check_rtpe
    }
}
```

Відповідно в checkrtpe.sh вносимо наступне:

```
#!/bin/bash
```

```
node01=10.0.2.11
node02=10.0.2.12
port=2225
return_code=1 # fail

# check remote instance fist
timeout 1 nc -z $node01 $port > /dev/null 2>&1
exit_status=$?
if [[ $exit_status -eq 0 ]]; then
  echo "ping successful to node01 [$node01]"
  exit $return_code
fi

# remote instance failed, check local
timeout 2 nc -z $node02 $port > /dev/null 2>&1
exit_status=$?
if [[ $exit_status -eq 0 ]]; then
  echo "ping successful to node02 [$node02]"
  return_code=0
fi

echo "return code [$return_code]"

exit $return_code
```

На обох серверах RTPE1 та RTPE2 робимо файл checkrtpe.sh виконавчим:

```
# chmod +x /etc/keepalived/checkrtpe.sh
```

Результатом виводу повинна бути відсутність помилок.

Далі виконуємо перезапуск утіліти keepalived на обох серверах RTPE1 та RTPE2 з метою використання нею нових налаштувань:

```
# systemctl restart keepalived
Та перевіряємо статус:
# systemctl status keepalived
```

```
root@ip-172-31-87-8:/home/ubuntu# systemctl status keepalived
o keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/usr/lib/systemd/system/keepalived.service; enabled; preset: enabled)
     Active: inactive (dead)
  Condition: start active            at Mon 2025-05-12 08:17:57 UTC; 3min 18s ago
            └─ ConditionFileNotEmpty=/etc/keepalived/keepalived.conf was not met
       Docs: man:keepalived(8)
             man:keepalived.conf(5)
             man:genhash(1)
             https://keepalived.org
```

В разі успіху маємо результат як на рисунку вище.

2.в. Встановлення ПЗ RTPengine.

Процедура виконання є аналогічною для серверів RTPE1 та RTPE2.

Завантажуємо допоміжні модулі:

```
# apt install -y build-essential git dpkg-dev debhelper libxtables-dev linux-headers-$(uname
-r) mc libevent-dev libcurl4-openssl-dev libpcap-dev libhiredis-dev libglib2.0-dev libpcre3-dev
libavcodec-dev libavformat-dev libavutil-dev libswresample-dev libnetfilter-queue-dev libjson-
glib-dev libbcg729-dev markdown libspandsp-dev libavfilter-dev libbencode-perl libcrypt-
```

openssl-rsa-perl libdigest-crc-perl libdigest-hmac-perl libnet-interface-perl libxmlrpc-core-c3-dev net-tools pkg-config unzip zlib1g-dev libssl-dev

```
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.10+dfsg-3ubuntu3.1) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-1024-aws
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-1028-aws.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart networkd-dispatcher.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-82-243:/home/ubuntu#
```

Загружаэмо ПЗ RTPengine з репозіторію git:

```
# git clone https://github.com/sipwise/rtpengine.git
```

```
root@ip-172-31-82-243:/home/ubuntu# git clone https://github.com/sipwise/rtpengine.git
Cloning into 'rtpengine'...
remote: Enumerating objects: 51303, done.
remote: Counting objects: 100% (935/935), done.
remote: Compressing objects: 100% (332/332), done.
remote: Total 51303 (delta 790), reused 686 (delta 599), pack-reused 50368 (from 3)
Receiving objects: 100% (51303/51303), 31.22 MiB | 30.59 MiB/s, done.
Resolving deltas: 100% (41110/41110), done.
root@ip-172-31-82-243:/home/ubuntu#
```

Заходимо в діректорію та компілюємо модуль ядра RTPengine:

```
# cd rtpengine/kernel-module && make && make install
```

```
/home/ubuntu/rtpengine/kernel-module/gen-rtpengine-kmod-flags >/home/ubuntu/rtpengine/kernel-module/rtpengine-kmod.mk
make -C /lib/modules/6.8.0-1024-aws/build M=/home/ubuntu/rtpengine/kernel-module O=/lib/modules/6.8.0-1024-aws/build modules
make[1]: Entering directory '/usr/src/linux-headers-6.8.0-1024-aws'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc-13 (Ubuntu 13.3.0-6ubuntu2-24.04) 13.3.0
  You are using:           gcc-13 (Ubuntu 13.3.0-6ubuntu2-24.04) 13.3.0
  CC [M]  /home/ubuntu/rtpengine/kernel-module/xt_RTPENGINE.o
  MODPOST /home/ubuntu/rtpengine/kernel-module/Module.symvers
  CC [M]  /home/ubuntu/rtpengine/kernel-module/xt_RTPENGINE.mod.o
  LD [M]  /home/ubuntu/rtpengine/kernel-module/xt_RTPENGINE.ko
  BTF [M] /home/ubuntu/rtpengine/kernel-module/xt_RTPENGINE.ko
Skipping BTF generation for /home/ubuntu/rtpengine/kernel-module/xt_RTPENGINE.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.8.0-1024-aws'
install -D xt_RTPENGINE.ko /lib/modules/6.8.0-1024-aws/updates/xt_RTPENGINE.ko
depmod -a
root@ip-172-31-82-243:/home/ubuntu/rtpengine/kernel-module#
```

Завантажуємо модуль ядра RTPengine та перевіряємо його статус:

```
modprobe xt_RTPENGINE && lsmod | grep RTPENGINE
```

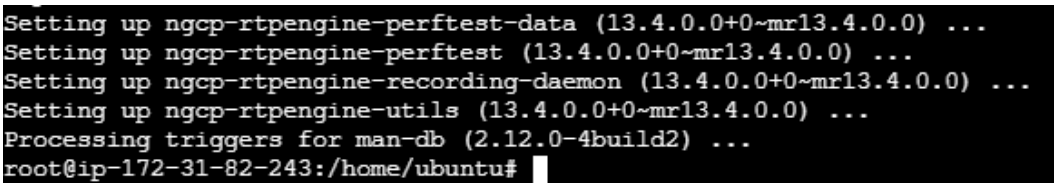
```
root@ip-172-31-82-243:/home/ubuntu/rtpengine/kernel-module# lsmod | grep RTPENGINE
xt_RTPENGINE           94208  0
x_tables               65536  2 xt_RTPENGINE,ip_tables
root@ip-172-31-82-243:/home/ubuntu/rtpengine/kernel-module#
```

Компілюємо підсистему користувача:

```
# cd .. && dpkg-buildpackage -b -us -uc
```

Виконуємо установку підсистеми користувача:

```
# cd ..
# dpkg -i ngcp-rtpengine-*.deb
```



Виконуємо налаштування конфігурації.
для RTPE1 в файл /etc/rtpengine/rtpengine.conf вносимо наступні правки:

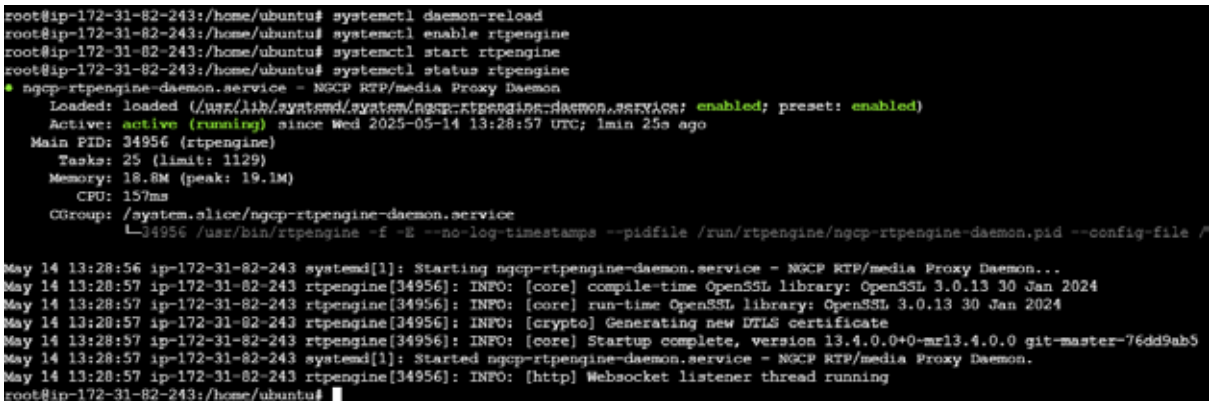
```
[rtpengine]
table = 0
interface = pub1/10.20.20.2;pub2/10.0.2.12
listen-ng = 10.20.20.2:2223
listen-http = 10.0.2.11:2225
listen-cli = localhost:2224

timeout = 60
silent-timeout = 3600
tos = 184
port-min = 30000
port-max = 40000
redis = 10.0.2.13:6379/1
subscribe-keyspace=2
redis-num-threads = 8
no-redis-required = true
redis-expires = 86400
redis-allowed-errors = -1
redis-disable-time = 10
redis-cmd-timeout = 0
redis-connect-timeout = 1000
для RTPE2 в файл /etc/rtpengine/rtpengine.conf вносимо наступні правки:
[rtpengine]
table = 0
interface = pub1/10.20.20.2;pub2/10.0.2.11
listen-ng = 10.20.20.2:2223
listen-http = 10.0.2.12:2225
listen-cli = localhost:2224
timeout = 60
silent-timeout = 3600
tos = 184
port-min = 30000
port-max = 40000
redis = 10.0.2.13:6379/2
subscribe-keyspace=1
redis-num-threads = 8
```

```
no-redis-required = true
redis-expires = 86400
redis-allowed-errors = -1
redis-disable-time = 10
redis-cmd-timeout = 0
redis-connect-timeout = 1000
```

Виконуємо запуск підсистеми користувача:

```
# systemctl daemon-reload
# systemctl enable rtpengine
# systemctl start rtpengine
# systemctl status rtpengine
```



3. Встановлення in-memory DB Redis

3.а. Налаштовуємо мережевий інтерфейс.

Для цього, в файл /etc/netplan/net.yaml вносимо наступні дані:

```
network:
    version: 2
    ethernets:
        enp0s3:
            dhcp4: false
            addresses: [10.0.2.15/24]
            nameservers:
                addresses:
                - 8.8.8.8
                - 8.8.4.4
                search: []
            optional: true

            routes:
              - to: default
                via: 10.0.2.1
                metric: 100
```

Де у лістингу:

- 10.0.2.15 – IP адреса серверу Redis
- 10.0.2.1 – IP адреса шлюзу.

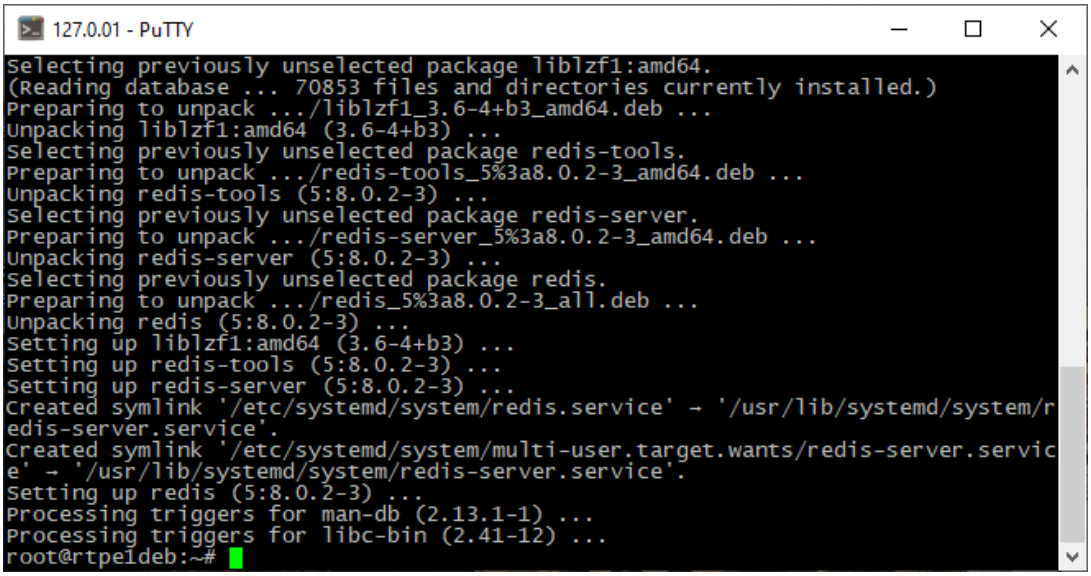
Для активації змін в конфігурації виконуємо команду:

```
# netplan apply
```

Результатом виводу повинна бути відсутність помилок.

3.б. Встановлюємо in-memory DB Redis:

```
# apt install redis
```



Вносимо правки в конфігураційний файл /etc/redis/redis.conf:

```
…
bind 10.0.2.15
notify-keyspace-events AKE
….
```

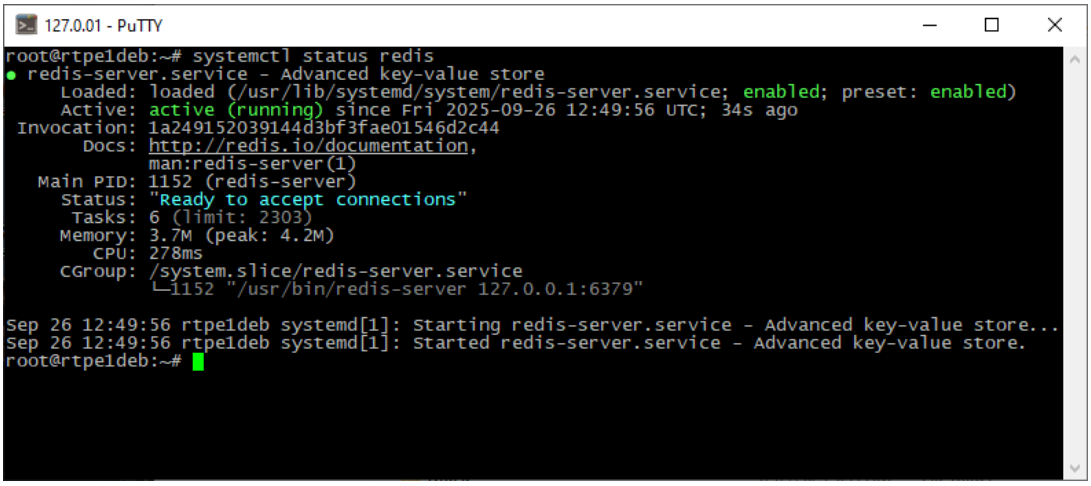
Виконуємо перезавантаження Redis:

```
# systemctl restart redis
```

Виконуємо перевірку статусу роботи Redis:

```
# systemctl status redis
```

У разі успіху результат буде виглядати як на зазначено нижче:



Результати виконання поставленої задачі.

Об'єктом дослідження є проведення двох експериментів:

А. Збереження голосового виклику та його коректне опрацювання після програмного або апаратного збою сигнального сервера КАМ1, який керує цим викликом.

Б. Збереження голосового виклику та його коректне опрацювання після програмного або апаратного збою медіа шлюза RTPE1, який опрацьовує голосовий потік.

Суть проведення експерименту *А* в наступному:

1) абонент Phone1 здійснює виклик абонента Phone2 в системі, наведеній на рис. 2;

2) після успішного встановлення з'єднання між цими абонентами ми штучно вносимо відмову сигнального серверу КАМ1, власне вимикаємо віртуальний комп'ютер;

3) оскільки всі SIP-транзакції "віддзеркалюються" в сигнальному сервері КАМ2, а ПЗ Keepalive впроваджує коректний статус сигнального сервера (в даному випадку КАМ2 змінює свій статус з резервного на основний) та продовжує керувати всіма транзакціями, раніше отриманими від КАМ1;

4) в разі отримання сервером КАМ2 сигнальних повідомлень в рамках SIP- транзакції, яка була створена КАМ1, КАМ2 відшукує цю транзакцію у власній БД та продовжує обробляти цей виклик.

Сигнальний трейс голосового виклику описаного вище, наведено на рис. 3, момент відповіді сигнального серверу КАМ2 позначено міткою [takeover].

Суть проведення експерименту *Б* в наступному:

1) абонент Phone1 здійснює виклик абонента Phone2 в системі, наведеній на рис. 2;

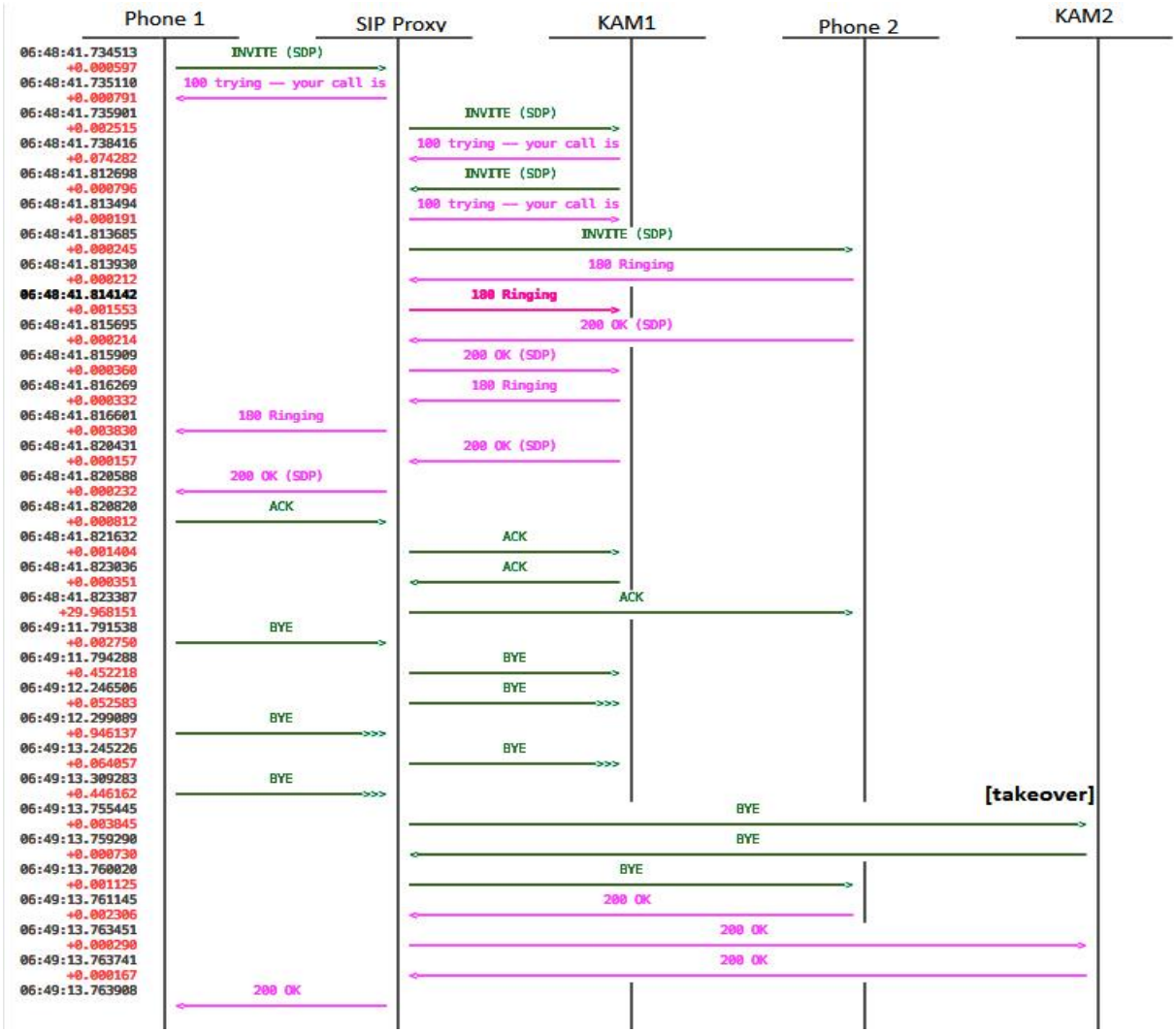


Рис. 2. Сигнальний трейс голосового виклика (Voice call signal trace) з подальшою відмовою сигнального сервера КАМ1

2) SIP транзакція буде опрацьована сигнальним сервером KAM1, а голосовий потік формується медіа шлюзом RTPE1;

3) після успішного встановлення з'єднання між цими абонентами ми штучно вносимо відмову сигнального серверу RTPE1, власне вимикаємо віртуальний комп'ютер;

4) REDIS відстежує вихід з ладу медіа шлюзу RTPE1 та виконує аварійне перемикання сесій на RTP2;

5) RTPE2 виконує обробку RTP трафіка для існуючих та нових голосових викликів.

На рис. 2 наведено голосовий виклик між абонентами Phone1 та Phone2 через високодоступну VoIP систему, яка складається з абонентських SIP-терміналів Phone1 та Phone2, серверів сигналізації KAM1 та KAM2 та SIP proxy, який в данному випадку виконує роль ретранслятора SIP повідомлень та надає можливість зібрати повний трейс сигнального обміну між вищезазначеними елементами системи. Розглянемо більш детально як саме проходить голосовий виклик.

На момент 06:48:41.734513 від абонента Phone1 надходить SIP-повідомлення **INVITE**, мета якого створити нову сесію голосового зв'язку з унікальним ідентифікатором **call-id** [1]. У відповідь на **INVITE**, абонент Phone1 отримує повідомлення **100 Trying** (запит у обробці) від SIP Proxy, який в свою чергу ретранслює повідомлення **INVITE** до сигнального серверу KAM1, що є активним за замовченням. KAM1 у свою чергу перевіряє внутрішню таблицю маршрутизації, знаходить IP адресу абонента Phone2, змінює його в полі «**To:**» повідомлення **INVITE**, та пересилає його до SIP Proxy в момент часу 06:48:41.812698.

Після отримання **INVITE**, SIP Proxy пересилає його кінцевому абоненту Phone2 та у відповідь отримує повідомлення **180 Ringing** (викликаємо абонента), яке в свою чергу транслюється сигнальному серверу KAM1.

В 06:48:41.815692 від абонента Phone2 до SIP Proxy надходить повідомлення **200 OK** (абонент відповів на виклик), яке далі транслюється на сигнальний сервер KAM1 і далі кінцевому абоненту Phone1. Отримавши повідомлення **200 OK** про те, що абонент прийняв голосовий виклик, Phone1 відсилає підтвердження у вигляді повідомлення **ACK,** яке в свою чергу транслюється кінцевому абоненту Phone2 через SIP Proxy та KAM1 (часовий інтервал 06:48:41.820820 - 06:48:41.823387).

Вносимо штучну відмову сигнального сервера KAM1 (вимикаємо віртуальну машину).

В 06:49:11.791538 абонент відправляє до SIP Proxy повідомлення про завершення виклику **BYE**, яке транслюється до сигнального серверу KAM1.

Не отримавши ніякої відповіді від сигнального серверу KAM1, SIP Proxy відправляє повідомлення **BYE** до серверу KAM2, який в свою чергу ідентифікує транзакцію по SIP call-id, раніше отриманого через DMQ інтерфейс від серверу KAM1 та пересилає його кінцевому абоненту Phone2.

В свою чергу абонент Phone2 відправляє повідомлення про підтвердження завершення виклику **200 OK** до абонента Phone1 по ланцюгу Phone2 – SIP Proxy – KAM2 – SIP Proxy – Phone1.

Таким чином ми бачимо, що після відмови сигнального серверу KAM1 SIP-діалог не обривається, а в 06:49:13.755445 продовжується оброблятися сигнальним сервером KAM2. Переключення на гарячий резерв виконується в автоматичному режимі, завдяки чому вплив на сервіс з'єднання відсутній.

Висновки. Розроблено та практично реалізовано високодоступну VoIP-систему, яка забезпечує безперервну роботу сервісу у разі відмов окремих компонентів. Запропоноване рішення цієї системи підтвердило свою ефективність під час експериментальної перевірки та може бути рекомендоване для практичного використання.

У статті детально розглянуто практичну реалізацію високодоступної VoIP-системи, яка складається з наступних основних компонентів:

– кластера SIP-серверів Kamailio;

– системи реплікації даних за допомогою Redis;

– кластера RTPengine для обробки медіапотоків;

– механізмів VRRP, BFD та Keepalived для контролю доступності вузлів.

Особливу увагу приділено:

– взаємодії між сигнальною та медіа-підсистемами;

– алгоритмам автоматичного перемикання при відмовах;

– збереженню безперервності сервісу для кінцевих користувачів.

У роботі наведено приклади конфігурацій, схеми взаємодії компонентів та результати тестування працездатності системи.

Наукова новизна роботи полягає у розробці та практичному обґрунтуванні комплексної архітектури високодоступної VoIP-системи, що базується на відкритому програмному забезпеченні.

Основні положення, які визначають наукову новизну, полягають у наступному:

– Запропоновано нову архітектурну модель високодоступної VoIP-системи, яка поєднує кластерну SIP-сигналізацію та відмовостійку медіаобробку.

– Удосконалено метод синхронізації SIP-реєстрацій та стану викликів шляхом використання in-memory бази даних Redis та механізмів обміну повідомленнями між вузлами Kamailio.

– Розроблено практичний механізм резервування RTP-медіашлюзів, що забезпечує збереження безперервності медіапотоків у разі відмови одного з вузлів.

– Доведено можливість побудови високодоступної VoIP-системи без використання комерційних рішень, що має важливе прикладне значення.

Отримані результати мають відтворюваний характер, що дозволяє застосовувати запропоновану архітектуру у реальних телекомунікаційних мережах.

Список літератури:

1. Mierla D.C., Modroiu E.R. “SIP Routing with Kamailio” 2022 356p.
2. RTPEngine. URL: https://rtpengine.readthedocs.io.
3. Carlson J.L. Redis in action. 2013. 320 p.
4. Keepalived. URL: https://keepalived.org.
5. Приймак С.О, Кравчук С.О. Метод підвищення доступності VoIP-систем. *Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки*. 2025. Т. 36(75). № 2. 123–129. DOI: https://doi.org/10.32782/2663-5941/2025.2.1/19
6. Pryimak S.O., Kravchuk S.O. VoIP system with high availability. *Information and Telecommunication Sciences.* 2025. *16*(1), 59–66. DOI: https://doi.org/10.20535/2411-2976.12025.59-66
7. Habraken J. Practical Cisco Routers. QUE USA, 1999. 315p.
8. Roy O.P., Kumar V. A Survey on Voice over Internet Protocol (VoIP) Reliability Research. *IOP Conference Series: Materials Science and Engineering.* 6th International Conference on Computers Management & Mathematical Sciences (ICCM 2020). 2020. Vol. 1020, pp. 12-19 Nirjuli, India, DOI https://doi.org/10.1088/1757-899X/1020/1/012015
9. Tulemu W., Design of an asterisk-based VoIP system and the implementation of security solution across the VoIP network. *World Journal of Advanced Research and Reviews*. 2024. Vol.1. № 23(01), P.875–906 DOI: https://doi.org/10.30574/wjarr.2024.23
10. Tobiloba K.A., Building a Secure Asterisk-Based VoIP System Design and Implementation, Researchgate, 2023. Vol. 4, № 9(82). p. 4–11. URL: https://www.researchgate.net/publication/388707610
11. Martin A., Gamess E., Urribarri D., Gómez J. A Proposal for A High Availability Architecture for VoIP Telephone Systems based on Open Source Software. *(IJACSA) International Journal of Advanced Computer Science and Applications,* 2021. Vol. 9, No. 9, 20. 2023. URL: https://thesai.org/Downloads/Volume9No9/Paper_1-A_Proposal_for_a_High_Availability_Architecture.pdf
12. The Sipwise C5 CE Handbook mr8.5.10. 400p. URL: https://www.sipwise.com/doc/mr8.5.10/handbook-ce.pdf.
13. Wu W. Packet Forwarding Technologies. Auerb Publications. 2008. 446 p.
14. Ahmed A., Madani H., Siddiqui T., Voip Performance Management and Optimization, Cisco Press. 2010. 448 p.
15. Rosenberg J., Schulzrinne H., Camarillo G., Johnston A., Peterson J., Sparks R., Handley M., Schooler E. SIP: Session Initiation Protocol. RFC 3261. Internet Engineering Task Force (IETF) 2002, 269p.
16. Pal S., Gadde R., Latchman H.A. On The Reliability of Voice Over IP Telephony. Computer Science, Engineering. 2011. URL: https://www.iiis.org /CDs2011/CD2011IMC/CCCT_2011/PapersPdf/TA224EV.pdf.
17. Vetoshko I.P., Kravchuk S.O. Possibilities of improving the voice services quality in 5G networks. *Information and Telecommunication Sciences*. 2023. Vol.14, No 2. pp. 9–16. https://doi.org/10.20535/2411-2976.22023.9-16

Priymak S.O., Kravchuk S.O. DEPLOYMENT OF HIGHLY AVAILABLE VOIP SYSTEM

The article describes the practical implementation of VoIP (Voice over IP) systems with increased availability, which provides voice communication over IP networks. An example of a practical implementation of a complete VoIP system is given, in particular, configuring and testing the main components to ensure service continuity, fault tolerance, scalability and resource efficiency.

The proposed approach for practical implementation of a VoIP system is based on the use of free software Kamailio (SIP signaling server), RTPengine (media traffic processing) and Redis (high-speed database for storing RTP transactions). All components are based on the Linux platform. The implemented architecture

includes mechanisms for automatic failover of signaling servers and media gateways using the VRRP protocol, load balancing and seamless switching to backup nodes in the event of failure of one or more components.

This is significantly different from traditional approaches, where the failure of one of the key elements (signaling server or media gateway) leads to the immediate termination of all active connections.

Experimental simulation of the system operation was performed, which confirmed its high efficiency: even in the event of a failure of 50% of the components, the system remains operational and does not interrupt any active voice call.

The results obtained may be useful for developers and operators of VoIP networks, mobile operators, as well as enterprises that using IP telephony for critical communications. The proposed approach to system implementation allows minimizing the risks of communication interruption, improving the quality of user service and ensuring high reliability of VoIP services.

Keywords: *VoIP system, SIP, Kamailio, RTP, RTPengine, VRRP, high availability, reliability.*

Дата першого надходження статті до видання: 22.01.2026
Дата прийняття статті до друку після рецензування: 20.02.2026
Дата публікації (оприлюднення) статті: 08.04.2026