ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

MẠNG MÁY TÍNH (THÍ NGHIỆM) (CO3094)

ASSIGNMENT 1 REPORT

DEVELOP A NETWORK APPLICATION

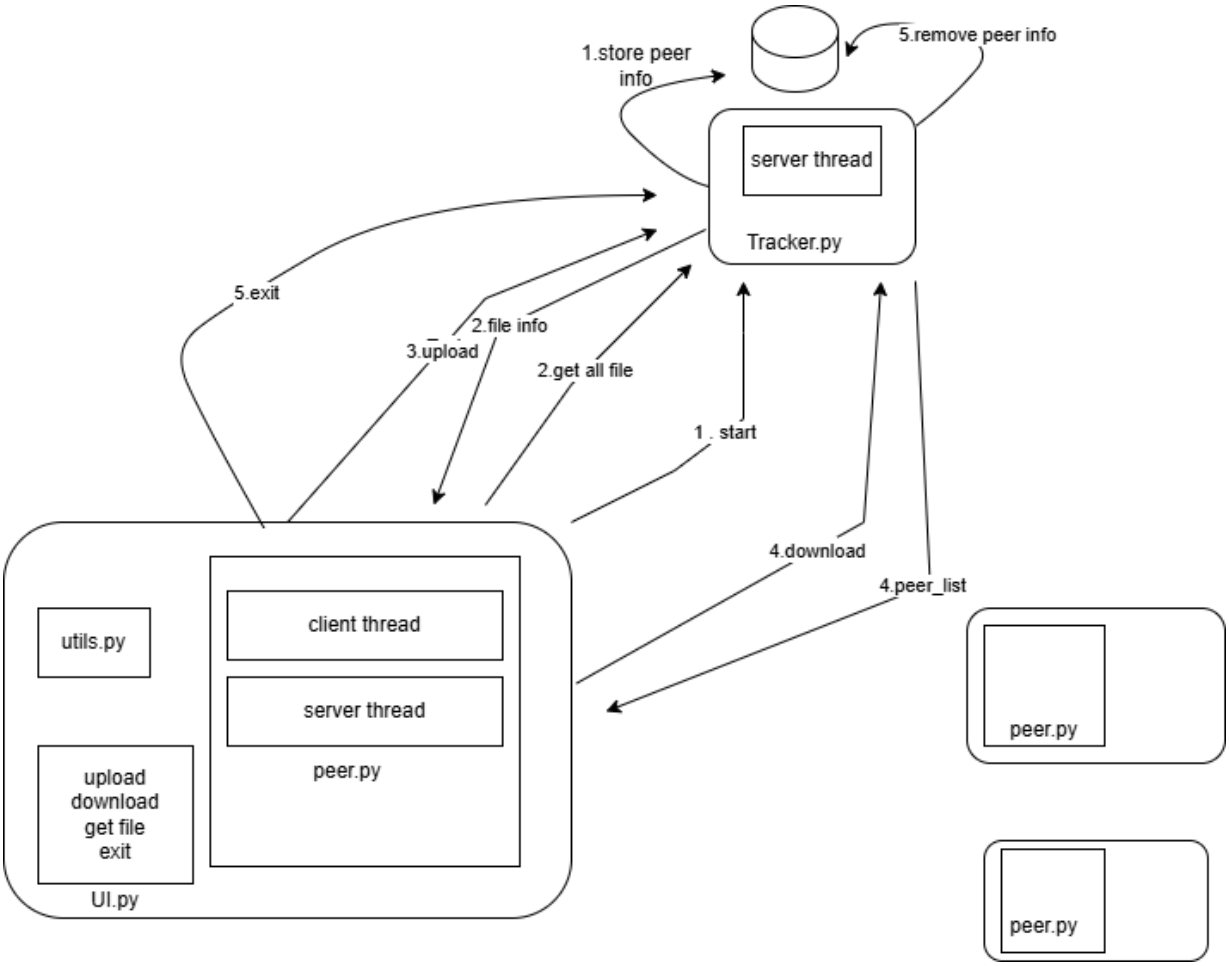
Giảng viên hướng dẫn: Nguyễn Phương Duy
Sinh viên: Đặng Trần Minh Nhật - 2212388
Trần Chính Bách - 2210187
Hoàng Nghĩa Toàn Phi - 2212541

TP. HỒ CHÍ MINH, THÁNG 11/2024

Mục lục

1 PHASE 1 : Define specific functions of the file-sharing application

1.1 Giai đoạn peer to tracker



- Tracker sẽ khởi động đầu tiên để nhận các request từ các peer.
- database kết nối với tracker sẽ dùng để chứa peerlist tương ứng với mỗi magnet text , và file .torrent
- Khởi động các peer bằng cách chạy UI.py
- peer sẽ gửi tín hiệu start đến tracker bằng HTTP GET.
- nếu peer này chứa file muốn thông báo cho các peer khác, tracker sẽ tiến hành lưu thông tin peer vào database.
- Lúc này sau khi đã start(được ký hiệu số 1) xong , người dùng có các lựa chọn như upload, dowload, get file, exit.
- Get file (được ký hiệu số 2)

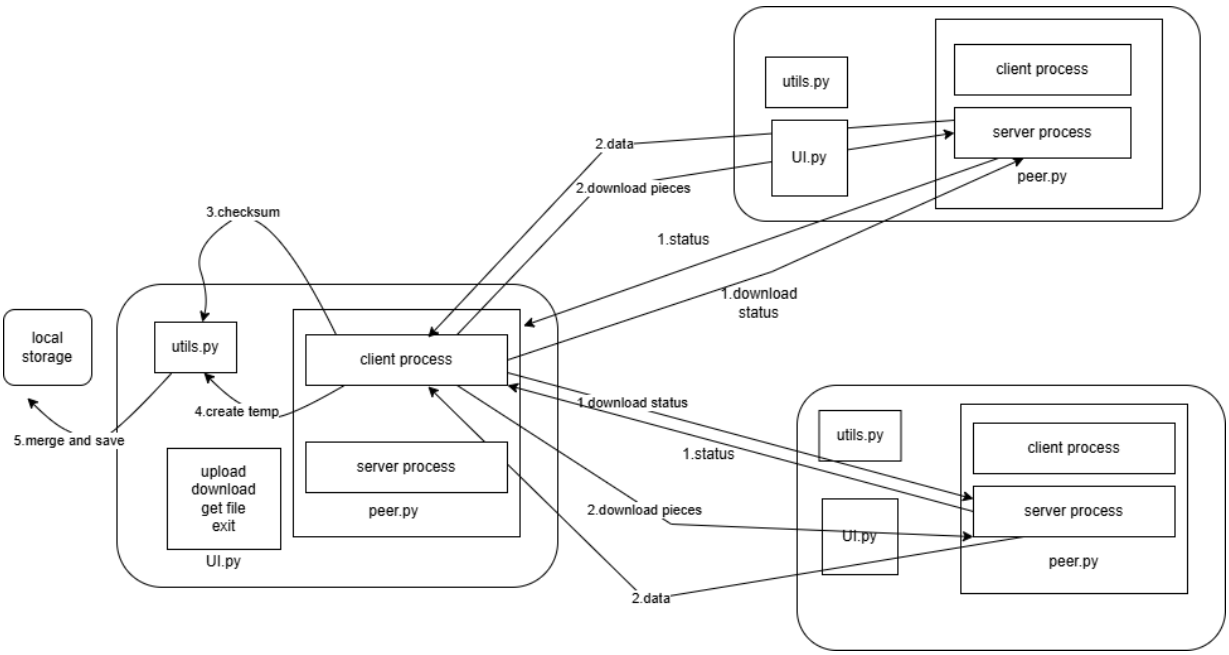
  - Khi người dùng không biết file nào tồn tại trong mạng có thể gọi get file
  - người dùng sẽ gửi HTTP GET đến tracker
  - Tracker trả về các magnet text hiện có (có thể lấy trường tên trong file .torrent tương ứng chứa trong database để trả về thêm).

- upload (được ký hiệu số 3)
  - Khi người dùng muốn thông báo với các peer khác trong mạng về file mình hiện có sẽ dùng chức năng này.
  - Chức năng sẽ gọi HTTP POST đến tracker server
  - phía tracker sẽ gọi lại store peer info

- Download (được ký hiệu số 4)
  - Khi người dùng đã có magnet text của file muốn download. Người dùng sẽ gọi hàm download và đưa vào magnet text.
  - Hàm sẽ gọi HTTP GET kèm tham số của magnet text tới tracker.
  - tracker sẽ trả về list peer (để người dùng biết địa chỉ của các peer khác), và .torrent file (để người dùng biết có bao nhiêu mảnh và hash của mỗi mảnh dùng cho việc check).

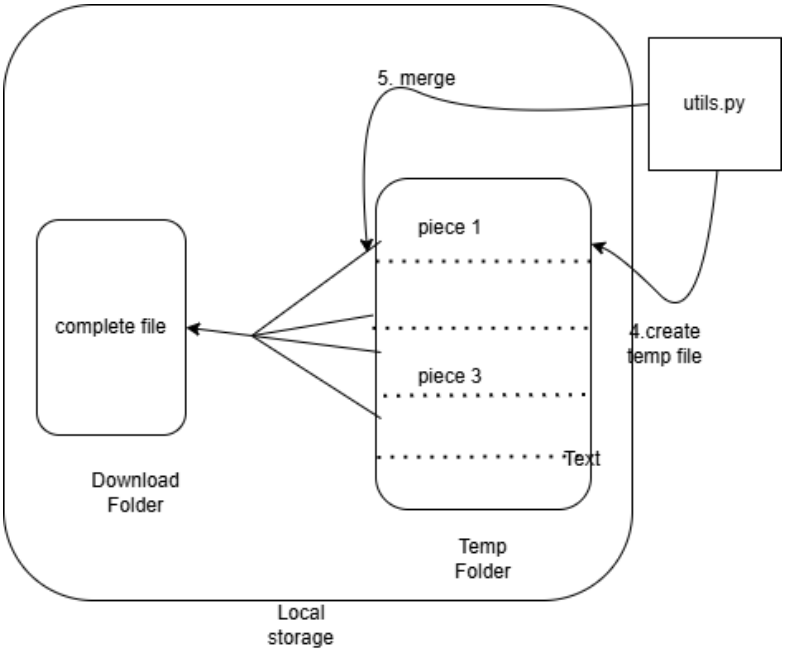
- Exit (được ký hiệu số 5)
  - Khi người dùng muốn thoát mạng, cần thông báo cho tracker để các peer khác không tốn thời gian tìm tới. Lúc này người dùng sẽ sử dụng exit.
  - Người dùng sẽ gửi HTTP POST đến tracker.
  - tracker sẽ gọi remove peer info để xóa thông tin của peer này trong các magnet text đang lưu.

1.2 Giai đoạn peer to peer



- Giai đoạn này chỉ xảy ra khi người dùng hoặc peer khác dùng chức năng download. Xảy ra ngay sau phần download của giai đoạn peer to tracker (lúc người dùng đã nhận được phản hồi thành công từ HTTP GET).
- Sau khi đã có peer list và torrent file, các bước sau đây sẽ được thực hiện tuần tự (được đánh số từ 1 đến 5):
- Trong mỗi peer sẽ có 2 phần , client process dùng đề nhận file khi người dùng gọi download và server process luôn lắng nghe để gửi file.
  - Dowload status : Người dùng gửi magnet text đến tất cả peer có trong list khác bằng TCP socket, các server process nghe và trả về các piece mà peer đó hiện có. Client từ đó tổng hợp ra list các mảnh với giá trị là thông tin các peer đang giữ mảnh đó.
  - Dowload pieces : dựa vào mản các status đã lấy ở bước trước, người dùng tiếp tục gửi thông tin bằng tcp socket đến các peer có chứa các mảnh, phía server process của các peer khác lắng nghe và gửi về các mảnh được yêu cầu của file .
  - Với từng file tải về xong, người dùng sẽ check xem thông tin có bị lỗi không bằng các giá trị hash tương ứng mỗi mảnh trong .torrent file mà tracker đã gửi.
  - Người dùng tạo temp folder để chứa các mảnh đã check.
  - Sau khi các mảnh đã đủ, gọi merge and save để lưu thành file hoàn chỉnh và chứa trong máy người dùng.

1.3 Mô tả ghép file



- Sau khi các client process lấy được data của mảnh được gửi bởi server process, các client process sẽ gọi tới hàm checksum trong trong utils.py, nếu đúng sẽ gọi hàm create temp để để file vào Temp Folder để chờ các client process khác tải các mảnh khác.
- Sau khi đã tải xong hết các mảnh, client process sẽ tiếp tục gọi merge trong utils để tiến hành gộp các file lạnh thành file hoàn chỉnh.

2 PHASE 2 : Implement and refine the application according to the functions and protocols defined in Phase 1

2.1 Tracker

Phía Tracker sẽ nhận các HTTP request nên cần phải xử lý dựa vào các HTTP request gửi tới

```
router.post("/start", async (req, res) => {
  const { peerIp, peerPort, magnetList } = req.body;
  try {
    // Lặp qua từng magnetText trong danh sách magnetList
    for (const magnetText of magnetList) {
      // Kiểm tra xem magnetText có tồn tại không
      let listPeer = await ListPeer.findOne({ magnetText });

      // Nếu không tồn tại, bỏ qua và tiếp tục vòng lặp
      if (!listPeer) continue;

      // Kiểm tra xem peer đã tồn tại trong list_peer chưa
      const peerExists = listPeer.list_peer.some(
        (peer) => peer.peerIp === peerIp && peer.peerPort === peerPor
      );

      // Nếu peer chưa tồn tại, thêm peer mới vào list_peer
      if (!peerExists) {
        listPeer.list_peer.push({ peerIp, peerPort });
        await listPeer.save(); // Lưu thay đổi vào database
      }
    }
    console.log("Connect to peer: ", peerIp, peerPort);
    res.status(200).json({ message: "Peers updated successfully." });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Hình 1: Api nhận req start

```
router.get("/getAllTorrents", async (req, res) => {
  try {
    const torrents = await Torrent.find(
      {},
      "magnetText metaInfo.name description"
    );

    // Chuyển đổi kết quả thành mảng các đối tượng với `filename` và `magnetText`
    const result = torrents.map((t) => ({
      filename: t.metaInfo.name, // Lấy tên file từ `metaInfo.name`
      magnetText: t.magnetText,
      description: t.description,
    }));
    console.log(result);
    res.status(200).json(result);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Hình 2: Api nhận yêu cầu trả về list các file có trong mạng

```
router.get("/download", async (req, res) => {
  const { peerIp, peerPort, magnetText } = req.body;
  try {
    const torrent = await Torrent.findOne({ magnetText });
    if (!torrent) {
      return res.status(404).json({ message: "There is no torrent" });
    }
    const listPeer = await ListPeer.findOne({ magnetText });
    let resList = [];
    if (!listPeer) {
      // Nếu không có listPeer, trả về torrent và danh sách peer rỗng
      res.status(200).json({
        torrent: torrent,
        listPeer: resList,
      });
    } else {
      // Tìm vị trí của peer trong list_peer
      const existingPeerIndex = listPeer.list_peer.findIndex(
        (peer) => peer.peerIp === peerIp && peer.peerPort === peerPort
      );
      // Tạo một bản sao của list_peer để trả về mà không bao gồm peer hiện tại
      resList = listPeer.list_peer.filter(
        (peer) => !(peer.peerIp === peerIp && peer.peerPort === peerPort)
      );
      if (existingPeerIndex === -1) {
        // Nếu peer chưa tồn tại, thêm peer mới vào list_peer trong database
        listPeer.list_peer.push({ peerIp, peerPort });
        await listPeer.save(); // Lưu thay đổi vào database
      }
      // Trả về torrent và listPeer không bao gồm peer hiện tại
      res.status(200).json({
        torrent: torrent,
        listPeer: resList,
      });
    }
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
```

Hình 3: Api nhận yêu cầu trả về listpeer có trong mạng đang giữ các mảnh file

```
router.post("/upload", async (req, res) => {
  const { peerIp, peerPort, Torrent: torrentData } = req.body;
  try {
    const { magnetText } = torrentData;
    let torrent = await Torrent.findOne({ magnetText });
    if (!torrent) {
      const newTorrent = new Torrent(torrentData);
      await newTorrent.save();
    } else {
      return res.status(409).json({ error: "already have on server" });
    }

    let listPeerOfTorrent = await ListPeer.findOne({ magnetText });

    if (!listPeerOfTorrent) {
      listPeerOfTorrent = new ListPeer({
        magnetText,
        list_peer: [{ peerIp: peerIp, peerPort: peerPort }],
      });
    } else {
      listPeerOfTorrent.list_peer.push({ peerIp, peerPort });
    }
    await listPeerOfTorrent.save();

    res.status(200).json({ message: "Torrent uploaded successfully." });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Hình 4: Api nhận yêu cầu ghi nhận peer gửi đang giữ các mảnh file

```
router.post("/exit", async (req, res) => {
  const { peerIp, peerPort } = req.body;
  try {
    await ListPeer.updateMany(
      {},
      { $pull: { list_peer: { peerIp, peerPort } } }
    );
    res.status(200).json({ message: "Peer removed successfully." });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Hình 5: Api nhận yêu cầu thoát của peer

2.2 Các peer

2.2.1 main.py, apiclient.py

File này có nhiệm vụ tạo ra các folder cần thiết để lưu file , lấy địa chỉ ip và gán port cho peer, lắng nghe input tương ứng. Mỗi input được nhập sẽ gọi sang apiclient.py để thực hiện . Phía apiclient sẽ chịu trách nhiệm gọi các làm logic trong peer.py.

```
command = input("Enter command number: ")

if command == "1":
    client.start()

elif command == "2":
    client.get_all_file()

elif command == "3":
    hashcode = input("Enter torrent hashcode: ")
    client.download(hashcode)

elif command == "4":
    print("Make sure your file is in folder Myfolder")
    filename = input("Enter filename: ")
    des = input("Enter description :")
    client.upload(filename,des)

elif command == "5":
    client.exit(host, port)
    break

else:
    print("Invalid command. Please try again.")
```

Hình 6: 5 Số tương ứng với 5 lệnh hợp lệ main lắng nghe

2.2.2 Peer.py

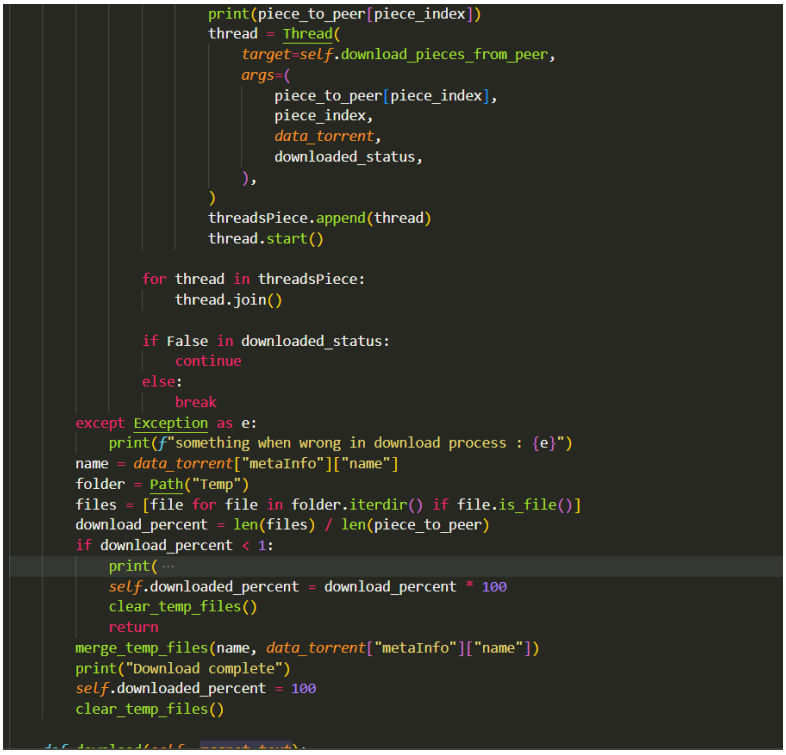
- Server process
  Khi start, chương trình sẽ dùng 1 thread chạy listen để lắng nghe , bất cứ kết nối nào tới lập tức tạo thêm 1 thread , truyền cho thread đó handle_listen để gửi cho các client process của các peer khác trạng thái các mảnh đang có hoặc dữ liệu của mảnh tùy thuộc vào input.

- Client process
  Được tạo khi peer sử dụng chức năng download, người dùng lúc này gửi yêu cầu lên server có tracker, tracker gửi về peerlist và .torent fil

  Lúc này hàm được gọi, client process sẽ tạo các thread , mỗi thread gửi dữ liệu yêu cầu các peer trong peerlist cập nhật thông tin về các mảnh hiện có , sau đó sẽ gửi yêu cầu đến các peer để lấy dữ liệu, sau đó mỗi thread gọi các hàm trong utils.py để check rồi bỏ vào Folder Temp , cuối cùng merge lại thành file hoàn chỉnh .

```python
def DownloadProcess(self, peerList, data_torrent):
    """Handle download process for each peer in the peerList"""

    # get status

    threadsStatus = []
    threadsPiece = []
    list_status = []
    lock = Lock()
    # Tạo thread cho mỗi peer
    for peer in peerList:
        thread = Thread(
            target=self.download_status_from_peer,
            args=(peer, data_torrent, list_status, lock),
        )
        threadsStatus.append(thread)
        thread.start()

    # Chờ tất cả các thread hoàn thành
    for thread in threadsStatus:
        thread.join()
    print(list_status)
    piece_to_peer = contruct_piece_to_peers(list_status)
    print(piece_to_peer)

    # {0: true,
    #  1: fale }

    downloaded_status = [False] * len(piece_to_peer)
    timeout = time.time() + 10
    # get piece
    try:
        while time.time() < timeout:
            # while True:
            for piece_index in piece_to_peer:
                if downloaded_status[piece_index] == False:
                    random.shuffle(piece_to_peer[piece_index])
                    print(piece_to_peer[piece_index])
```

Hình 7: client process của chương trình(1)



```
                    print(piece_to_peer[piece_index])
                    thread = Thread(
                        target=self.download_pieces_from_peer,
                        args=(
                            piece_to_peer[piece_index],
                            piece_index,
                            data_torrent,
                            downloaded_status,
                        ),
                    )
                    threadsPiece.append(thread)
                    thread.start()

                for thread in threadsPiece:
                    thread.join()

                if False in downloaded_status:
                    continue
                else:
                    break
            except Exception as e:
                print(f"something when wrong in download process : {e}")
            name = data_torrent["metaInfo"]["name"]
            folder = Path("Temp")
            files = [file for file in folder.iterdir() if file.is_file()]
            download_percent = len(files) / len(piece_to_peer)
            if download_percent < 1:
                print(...
                self.downloaded_percent = download_percent * 100
                clear_temp_files()
                return
            merge_temp_files(name, data_torrent["metaInfo"]["name"])
            print("Download complete")
            self.downloaded_percent = 100
            clear_temp_files()
```

Hình 8: client process của chương trình(2)

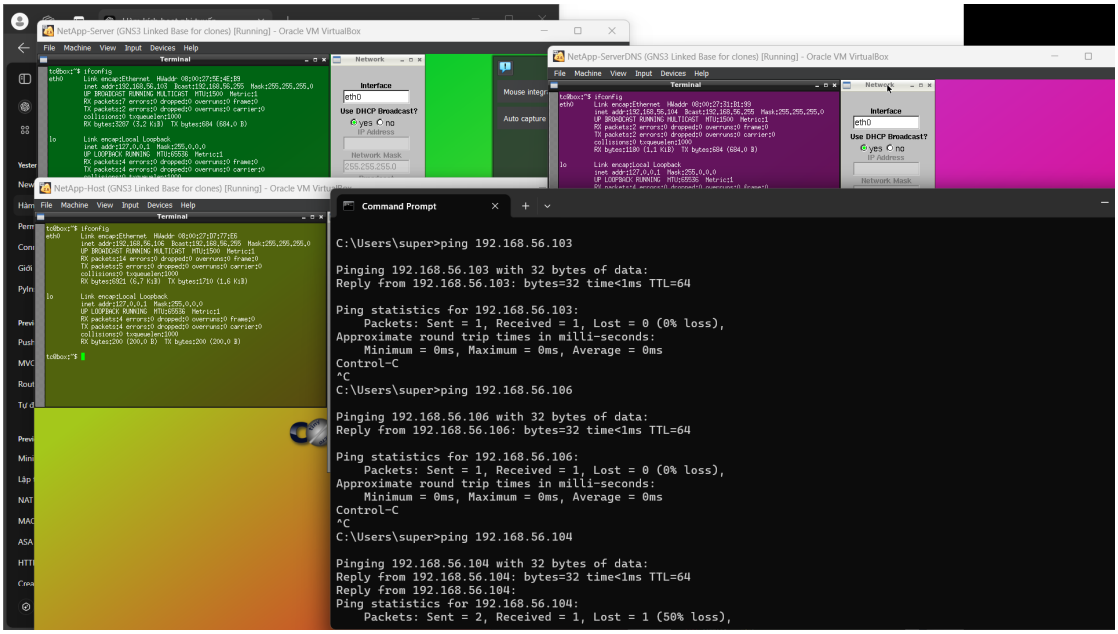
2.3 Kiểm tra/ chạy thử

Vì tracker đã deploy trên render để nhận HTTP, nhóm sẽ tiến hành test các peer bằng máy ảo.

2.3.1 Tạo môi trường

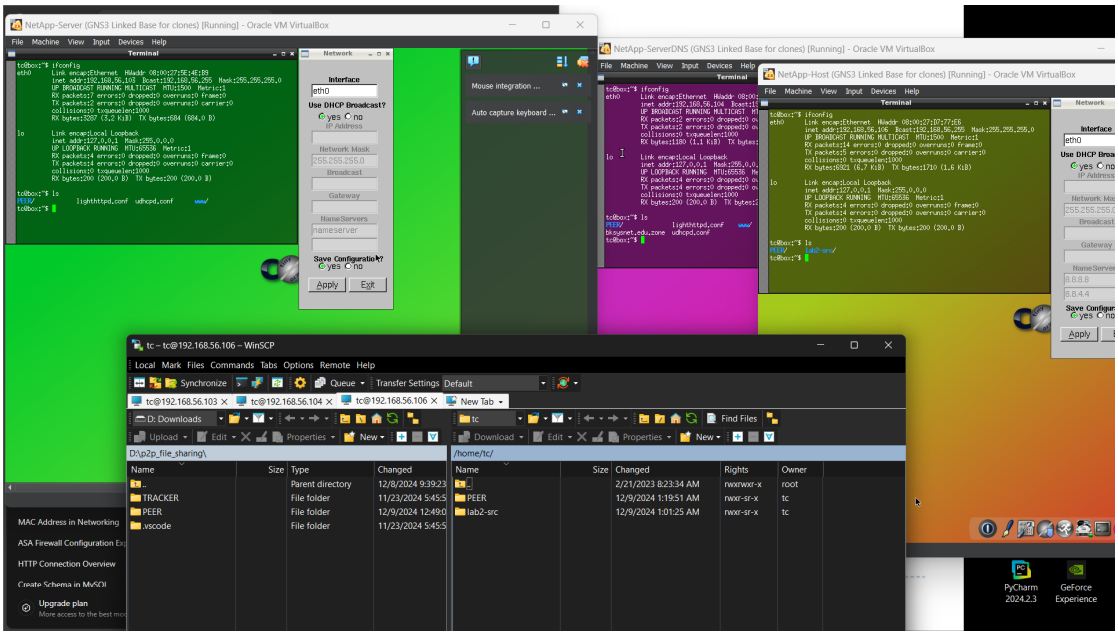
Các bước để chuẩn bị chạy ứng dụng trên máy ảo gồm các bước sau đây:

- Khởi động các máy ảo bất kì (nhóm khởi động 3 máy NetApp host, DNS và HTTP)
- Vào phần network chọn Host only, vào control panel chọn network chọn use DHCP Broadcast, ping thành công giữa máy vật lý và các máy ảo



Hình 9: ping các máy vật lí tới các mấy ảo

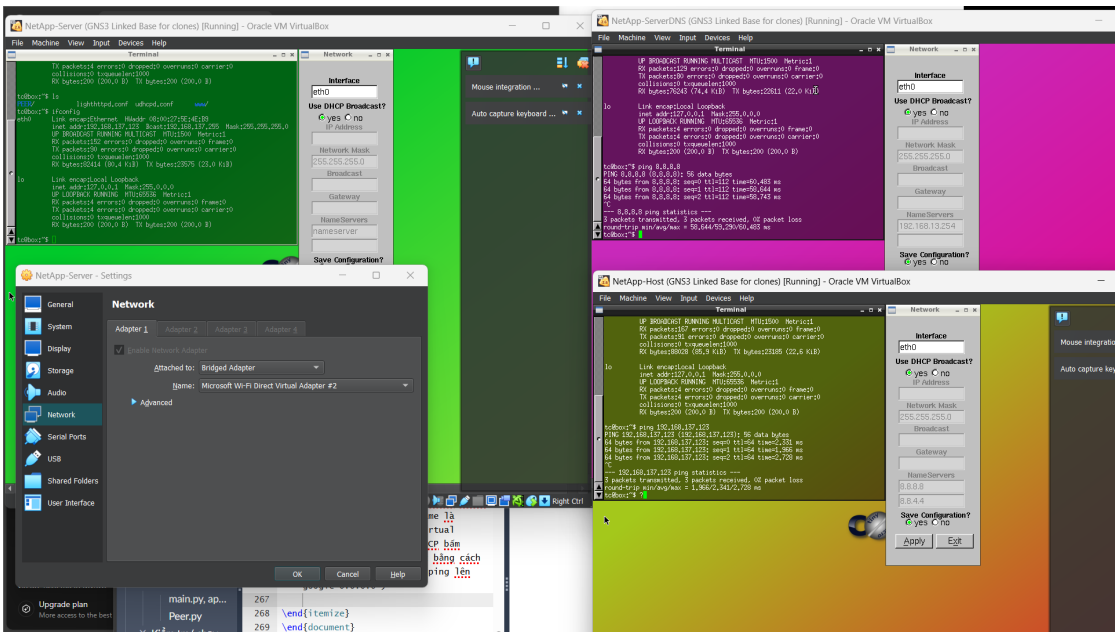
- Dùng winscp chuyển folder PEER trong thư mục p2p_file_sharing vào các máy ảo



Hình 10: chuyển folder vào máy ảo thành công

- Vào phần network chuyển thành Bridged Adapter, chọn name là Microsoft Wifi Direct

Virtual Adapter. Vào phần use DHCP bấm apply lại (có thể check bằng cách ping máy ảo với nhau và ping lên google 8.8.8.8)

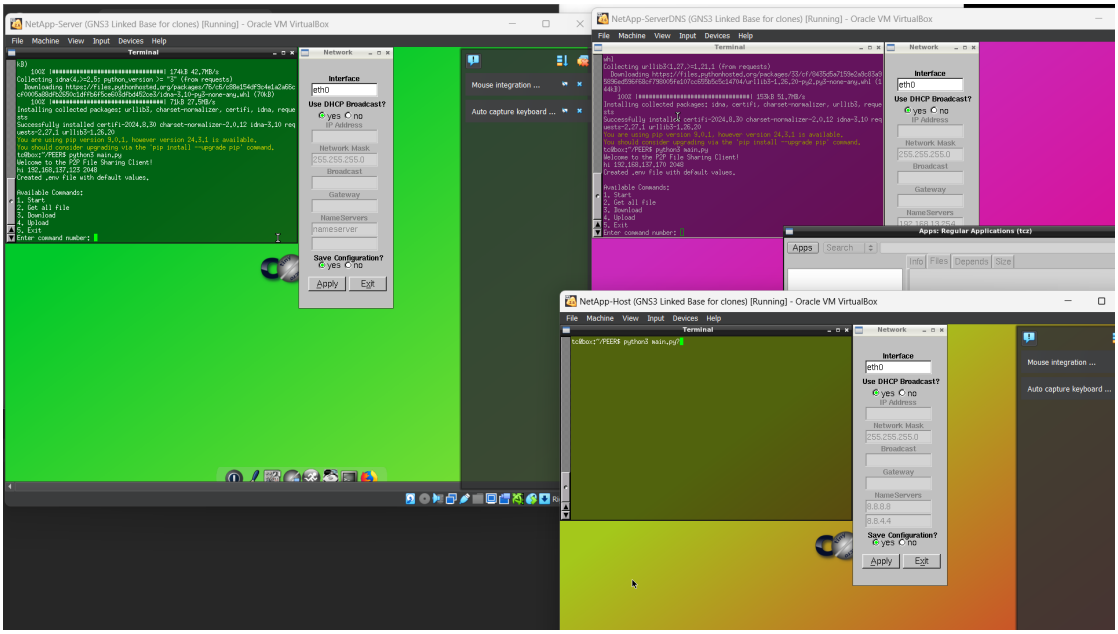


Hình 11: ping google, máy ảo dùng network bridge adapter

Sau các bước trên các máy ảo đã có Folder peer, vì nhóm gửi requests qua HTTP tới tracker nên cần thư viện requests, để tải thư viện cần thực hiện các bước sau:

- Ở mỗi máy ảo, nhập câu lệnh

```
sudo pip3 install requests --trusted-host=pypi.python.org \
    --trusted-host=pypi.org --trusted-host=files.pythonhosted.org
```

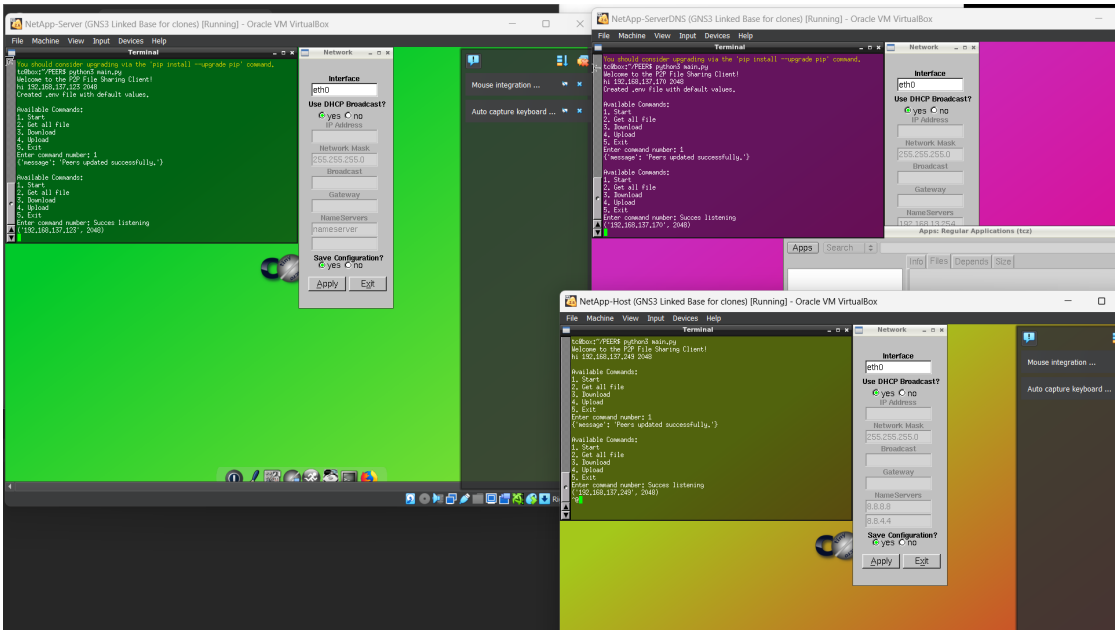


Hình 12: tải thư viện requests về máy ảo

Sau bước này,chương trình đã có thể chạy bằng cách di chuyển đến thư muc /PEER và dùng lệnh

```
python3 main.py
```

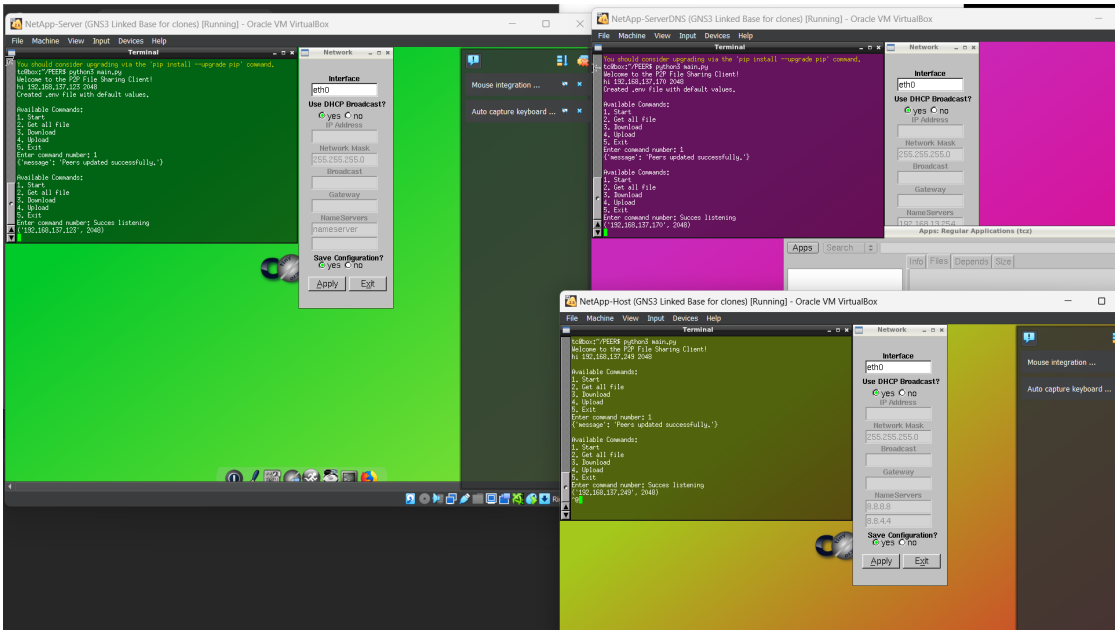
để chạy.



Hình 13: chạy chương trình thành công

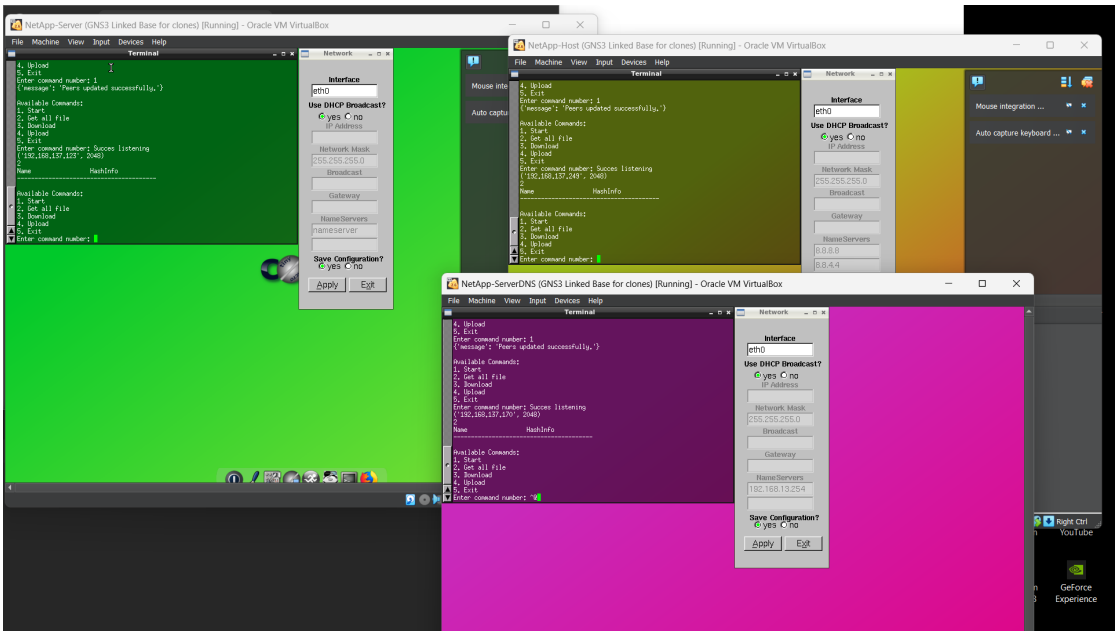
2.3.2 Kiểm tra các tính năng

- Tính năng start(số 1): khi chạy peer sẽ khởi tạo các server proccess (bắt đầu listening)



Hình 14: start thành công

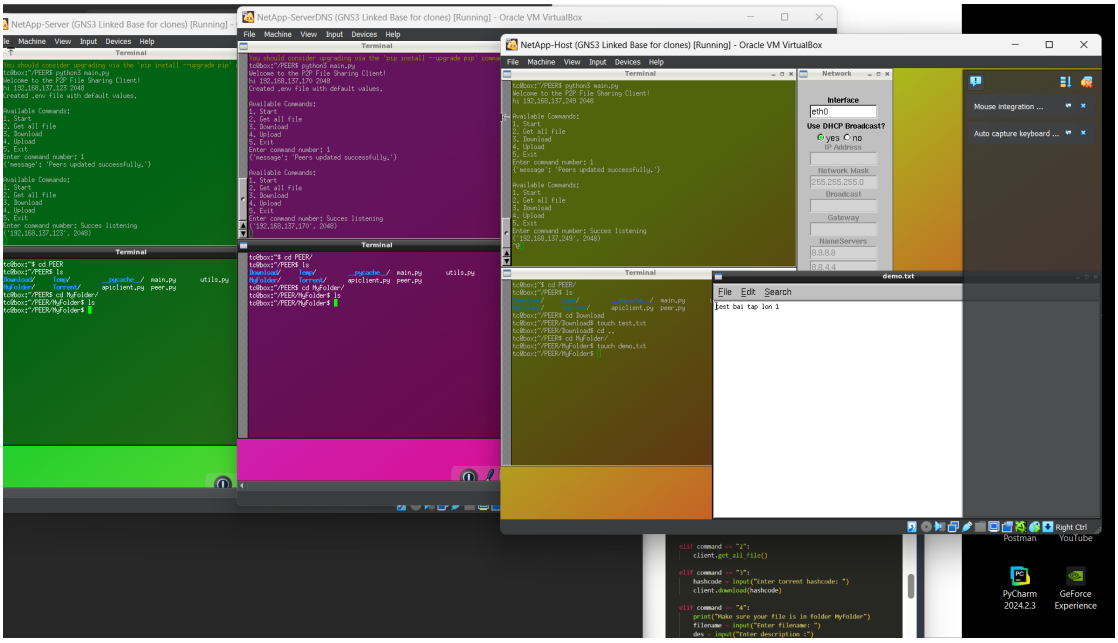
- Tính năng Get all file (số 2): Peer gửi HTTP request lên server và server trả về danh sách tên file và magnet link tương ứng



Hình 15: get all file thành công

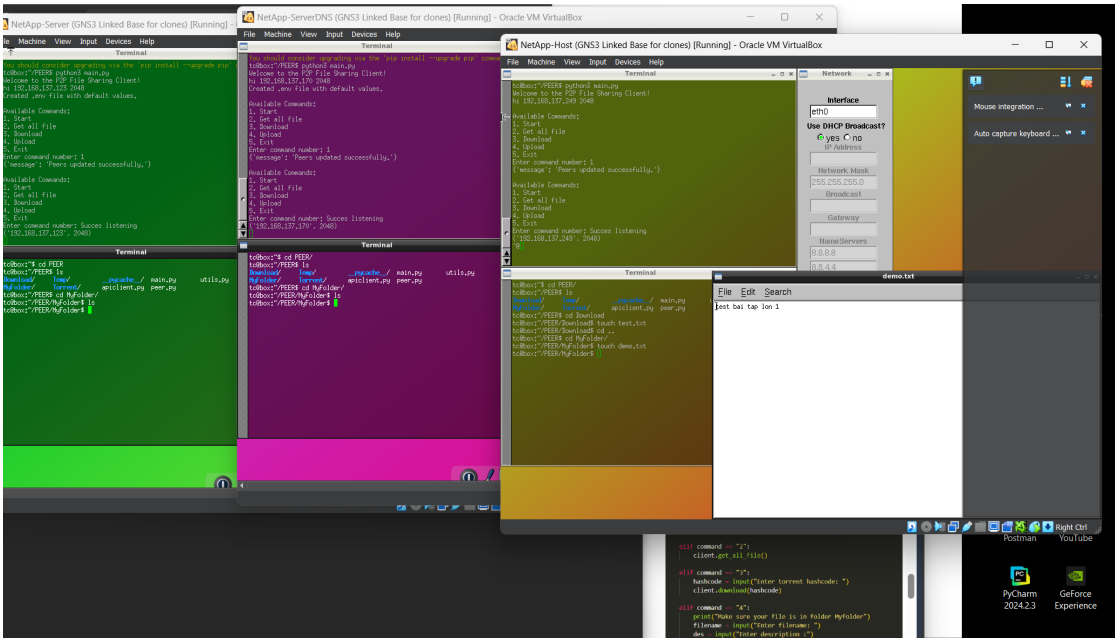
vì chưa có ai upload file gì hết lên danh sách trả về rỗng, cần test lại lần nữa khi đã upfile.

- Tính năng Upload (số 4) : tạo 1 file txt trong MyFolder sau đó gọi lên số 4.



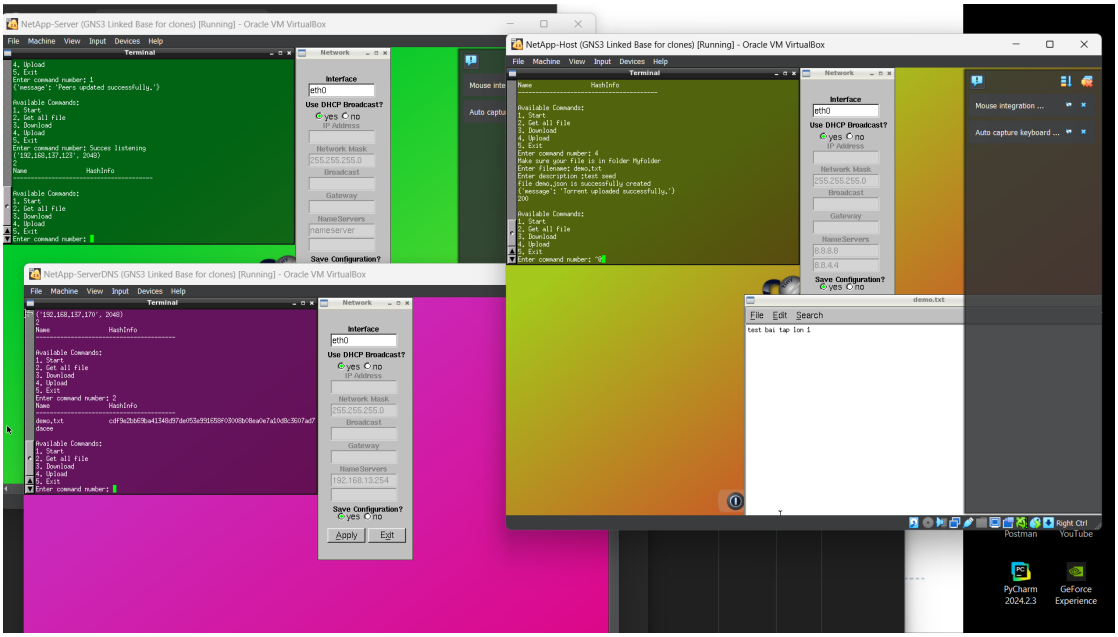
Hình 16: tạo test.txt trong MyFolder

Dùng lệnh số 2 ở máy khác để kiểm tra upload thành công



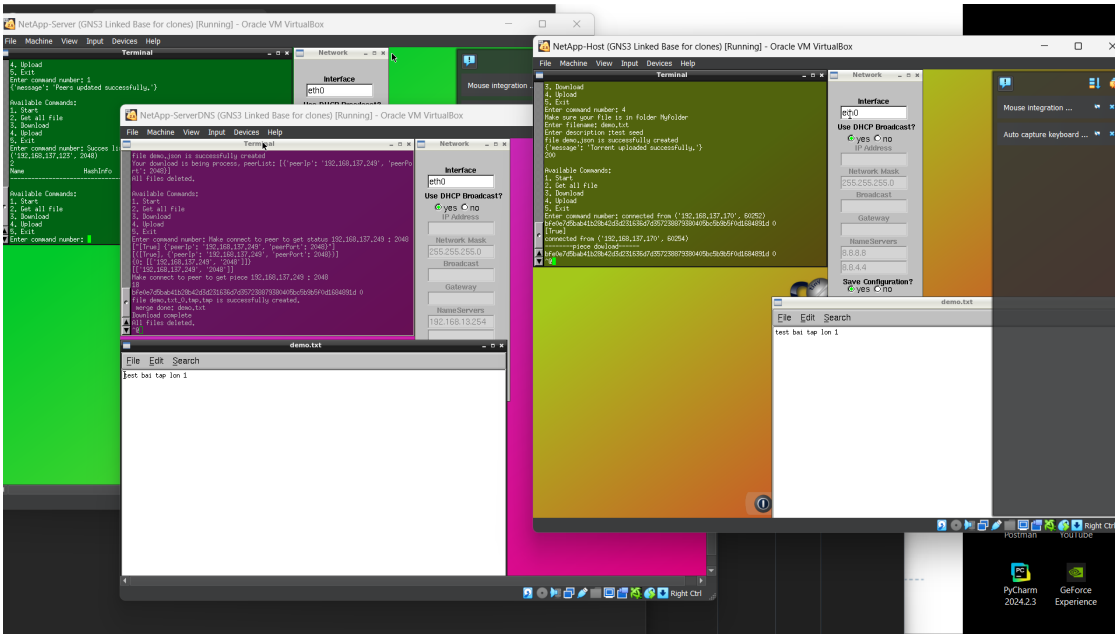
Hình 17: tạo test.txt trong MyFolder, chuẩn bị upload bằng lệnh 4

‘



Hình 18: Lệnh 2 ở máy DNS đã nhận thấy file mới

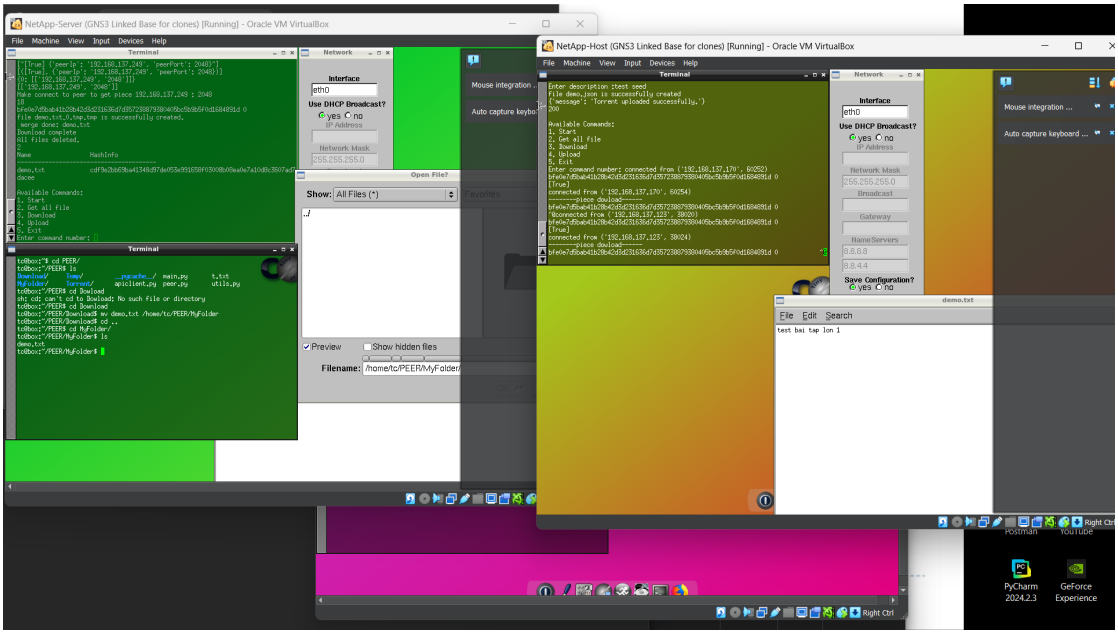
- Tính năng Download (số 3) : Người dùng copy hoặc nhập magnet link tìm được ở lệnh 2 vào để download file



Hình 19: Download test.txt thành công, màn hình netApp Host hiển thị có connect

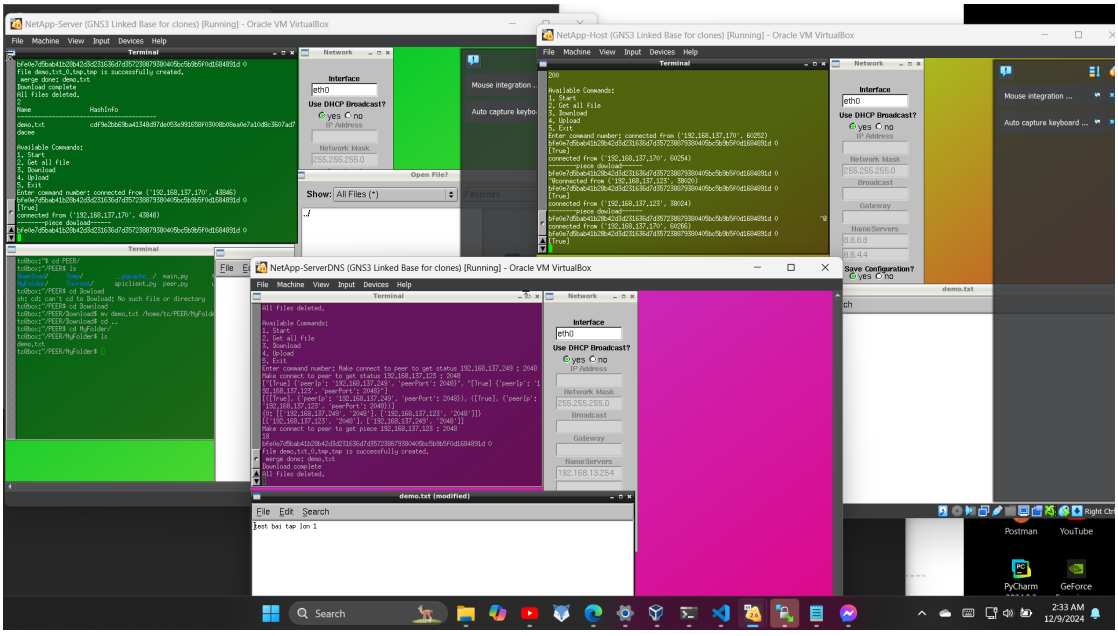
- Tính năng download từ nhiều máy khác nhau:

– Dem file đã đã tải được ở Download vào MyFolder.



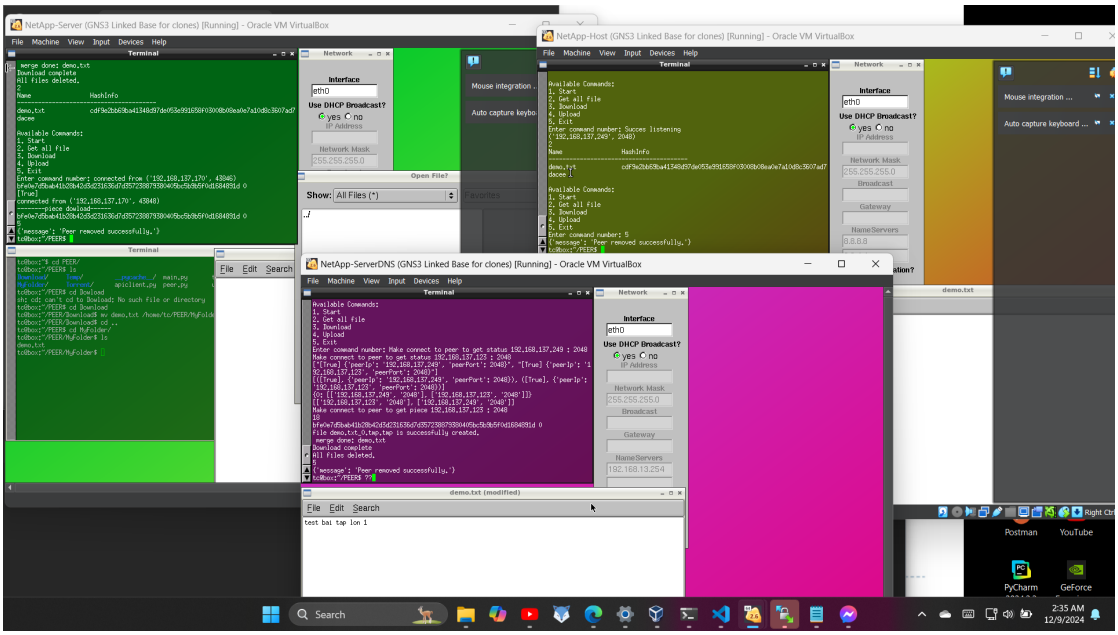
Hình 20: chuyển test.txt vào myFolder

– gọi Download xem 2 máy còn lại có nhận kết nối không.



Hình 21: 2 máy host và Http server đều nhận kết nối từ DNS để tải file, download thành công

- Chức năng exit (số 5) : gọi đến server , tracker sẽ bỏ ip và port của người dùng ra khỏi peerlist.



Hình 22: Thoát thành công