SKRIPSI

PENELITIAN PENGARUH MAC ADDRESS COLLISION PADA PERANGKAT YANG DAPAT MERUBAH MAC ADDRESS

GABRIEL POSSENTI KHEISA DRIANASTA

19/442374/PA/19123

PROGRAM STUDI S1 ELEKTRONIKA DAN INSTRUMENTASI

DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS GADJAH MADA

YOGYAKARTA

2022

HALAMAN PENGESAHAN SKRIPSI

PENELITIAN PENGARUH MAC ADDRESS COLLISION PADA PERANGKAT YANG DAPAT MERUBAH MAC ADDRESS

Diusulkan oleh

GABRIEL POSSENTI
19/442374/PA/19123

Telah disetujui
pada tanggal 3 Juni 2022
Pembimbing

Bakhtiar Alldino A.S., S.Si, M.Cs	Prof. Dr. Ir. Jazi Eko Istiyanto, M.Sc.
Pembimbing I	Pembimbing II

Ahmad Ashari, Dr-tech

Penguji

PERNYATAAN

Dengan ini saya menyatakan bahwa Laporan Skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaan di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Yogyakarta, tanggal-bulan-tahun

Gabriel Possenti Kheisa Drianasta

PRAKATA

Puji dan syukur ke hadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul “Penelitian Pengaruh Mac Address Collision Pada Perangkat Yang Dapat Merubah Mac Address”. Skripsi ini disusun dalam rangka pemenuhan salah satu syarat penyelesaian Program Studi Elektronika dan Instrumentasi, Departemen Ilmu Komputer dan Elektronika, Universitas Gadjah Mada

Penulis menyadari bahwa skripsi ini dapat selesai karena adanya bimbingan, bantuan, dukungan, dan nasehat dari berbagai pihak. Pada kesempatan ini, penulis ingin menyampaikan terimakasih setulus-tulusnya kepada:

1. Bakhtiar Alldino A.S., S.Si, M.Cs selaku pembimbing I atas segala bimbingan, arahan, dan sarannya yang berarti dalam penyelesaian skripsi ini;
2. Prof. Dr. Ir. Jazi Eko Istiyanto, M.Sc., IPU., ASEAN Eng. selaku pembimbing II atas segala bimbingan, arahan, dan sarannya yang berarti dalam penyelesaian skripsi ini;
3. Aufaclav Zatu Kusuma Frisky, S. Si, M. Sc. selaku Dosen Pembimbing Akademik atas segala bimbingan, terutama dalam hal sistem akademis.
4. Rekan – rekan Mahasiswa Elektronika dan Instrumentasi, atas kerjasama dan bantuannya;
5. Seluruh pihak yang terlibat hingga penulis tidak bisa sebutkan satu per satu.

DAFTAR ISI

INTISARI
PENELITIAN PENGARUH MAC ADDRESS COLLISION PADA PERANGKAT YANG DAPAT MERUBAH MAC ADDRESS

Oleh

Gabriel Possenti
19/442374/PA/19123

Alamat *Media Access Control* (MAC) atau MAC *address* adalah suatu alamat dengan panjang 48 bit yang merupakan alamat unik yang diberikan untuk perangkat *Network Interface Card* (NIC). Setiap perangkat dapat diidenfikasi secara unik pada suatu jaringan karena memiliki MAC *address* yang berbeda – beda. MAC *addresss*, sebagai alamat fisik, terdapat pada lapisan kedua dari *Open Systems Interconnection* (OSI) *layer* atau disebut juga *Data Link Layer*, yang dimana pada layer ini setiap bingkai atau *frame* data dikemas dengan MAC *address* sumber dan MAC *address* tujuan untuk menandakan pengiriman dan penerimaan data di jaringan, serta memungkinkan perangkat bertukar data dengan benar.

MAC *address collision* terjadi ketika dua atau lebih perangkat di suatu jaringan menggunakan MAC *address* yang sama. Beberapa sistem operasi terbaru, seperti Windows 10 dan Android 6.0, mendukung penggantian MAC *address* secara acak untuk meningkatkan privasi pengguna dengan menyembunyikan MAC *address* yang sesungguhnya. Jika dua perangkat di suatu jaringan memiliki MAC *address* yang sama, maka jaringan tidak dapat membedakannya, dan dapat menyebabkan perselisihan dalam transmisi data.

Penelitian Pengaruh MAC address collision dilakukan pada secara langsung pada suatu router fisik dan dilakukan juga pada *virtualized environment* pada VirtualBox.

ABSTRACT

RESEARCH ON THE EFFECT OF MAC ADDRESS COLLISION ON DEVICES THAT CAN CHANGE MAC ADDRESS

By

Gabriel Possenti

19/442374/PA/19123

Media Access Control (MAC) address or MAC address is an address with a length of 48 bits which is a unique address given to Network Interface Card (NIC) devices. Each device can be uniquely identified on a network because it has a different MAC address. MAC addresses, as physical addresses, are at the second layer of the Open Systems Interconnection (OSI) layer or also called the Data Link Layer, where at this layer each data frame is packed with a source MAC address and destination MAC address to indicate sending and receiving data on network, and allows devices to exchange data properly.

MAC address collision occurs when two or more devices on a network use the same MAC address. Some of the latest operating systems, such as Windows 10 and Android 6.0, support changing the MAC address randomly to increase user privacy by hiding the real MAC address. If two devices on a network have the same MAC address, the network cannot tell the difference, and can cause disputes in data transmission.

Research on the effect of MAC address collision was carried out directly on a physical router and also carried out in a virtualized environment on VirtualBox.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam suatu jaringan internet, terdapat aturan - aturan atau standar agar suatu perangkat dapat berkomunikasi satu sama lain. Salah satu standar yang paling umum adalah standar IEEE 802 yang umum digunakan dalam jaringan Wi-Fi. (IEEE 802, 2004) Di dalam IEEE 802, terdapat dua lapisan, yakni lapisan Data Link Layer yang terdiri dari *Logical Link Layer (LLC)* dan *Medium Access Control* (MAC), dan *Physical Layer*. MAC adalah lapisan yang mengatur pertanggungjawaban perangkat keras dalam komunikasi antar kabel, antar optik, atau nirkabel. MAC address mengatur pengenalan paket internet, pengalamatan tujuan paket, dan kontrol akses dalam medium fisik. Alamat MAC pada setiap perangkat unik, umumnya tidak dapat diubah dan memiliki panjang data 48-bit, dimana 24-bit pertama adalah kode dari perusahaan penerbit perangkat.

Ketika dua atau lebih perangkat di suatu jaringan memiliki MAC address yang sama, dapat membuat perselilishan pada lapisan *Open Systems Interconnection* (OSI) kedua, atau *Data Link Layer*. Kejadian ini disebut juga MAC *address collision*, dan dapat menyebabkan konflik dalam proses transmisi data dengan pengemasan alamat MAC sumber dan alamat MAC tujuan. Sebelum perangkat ingin mengirim paket data ke perangkat lain pada suatu jaringan, perangkat tersebut harus mengetahui dahulu alamat MAC perangkat yang dituju agar paket data dapat dikirim dengan tujuan yang benar. Proses pemetaan ini dilakukan oleh *Address Resolution Protocol* (ARP), yaitu sebuah protokol yang digunakan untuk menemukan alamat MAC tujuan dengan mencari terlebih dahulu alamat IP atau *Internet Protocol address* dari perangkat tujuan. Ketika ARP menemukan alamat IP perangkat tujuan, maka perangkat pengirim mengirim permintaan ARP ke jaringan tersebut untuk meminta alamat MAC yang sesuai. Hubungan antara alamat MAC dan ARP adalah jika alamat MAC adalah suatu kode untuk mengidentifikasi perangkat pada suatu jaringan, maka ARP merupakan

protokol untuk menemukan alamat MAC antara perangkat pengirim dan penerima melalui pencarian alamat IP, dimana alamat IP ini sendiri berjalan pada lapisan yang lebih tinggi pada lapisan OSI ke-3, atau disebut juga *Network Layer*. Layer ke-3 ini bertanggung jawab dalam pengiriman paket data antar jaringan, dimana setiap perangkat memiliki alamat IP yang unik agar dapat diidentifikasi oleh jaringan, yang memiliki hubungan erat dengan ARP, dimana ARP itu sendiri memetakan alamat IP ke alamat MAC yang berjalan pada lapisan OSI satu tingkat dibawah IP. *Network Layer* juga menangani proses *routing*, yaitu proses mengirimkan paket data melalui jalur yang optimal. *Network Layer* mampu menentukan jalur terbaik untuk mengirimkan data dengan adanya informasi *routing* yang tersimpan pada *routing table*. Dalam hal ini, *Internet Protocol* (IP) juga merupakan elemen yang penting, karena IP, ARP, dan MAC berhubungan erat dan memiliki persamaan untuk mengidentifikasi perangkat pada suatu jaringan.

1.2 Rumusan Masalah

Meskipun MAC Address umumnya tidak dapat diubah, namun beberapa perangkat bahkan manufaktur boleh merubah alamat yang sudah bersifat hard-coded tersebut. Dengan mengubah MAC Address, identitas perangkat secara fisik juga berubah. (Cardenas, Edgar D., 2013) Dengan mengubah 24-bit pertama, maka perangkat tersebut dapat memanipulasi kode penerbit perangkat yang sebenarnya. Beberapa penerbit perangkat tersebut, khususnya dengan sistem operasi (OS) dan dukungan driver *Network Interface Card* (NIC) terbaru yang memperbolehkan mengubah MAC address demi tujuan anonimitas penggunanya. Tetapi jika MAC address perangkat tersebut memiliki address yang sama dengan perangkat lain dalam suatu jaringan yang sama, maka secara teoritis akan menimbulkan masalah pada jaringan tersebut.

Sebagai tambahan, metode *randomization* alamat MAC dapat dilakukan dengan menggunkanan fungsi *hash* dari alamat MAC asli, sehingga tercipta alamat

MAC baru yang berbeda dari MAC address perangkat tersebut. (Jean-François Determe, Sophia Azzagnuni, François Horlin, Philippe De Doncker, 2022). Metode *randomization* alamat MAC dapat juga dilakukan dengan menambahkan *salt*, yaitu nilai acak atau *random* yang ditambahkan pada suatu nilai yang ingin di-*hash*. (Junade Ali, Vladimir Dyo, 2020). Meskipun fungsi *salt* ini pada praktiknya untuk mengamankan password dari peretas yang mengetahui nilai *hash* yang umum, *salt* juga dapat mengurangi kemungkinan *collision* dengan lebih baik.

Selain anonimitas dan privasi, pengguna menggunakan fitur tersebut untuk melewati filter MAC, dimana pada beberapa kasus *administrator* jaringan membatasi akses ke jaringan tersebut baik dengan skema *blacklist*, dimana perangkat dengan MAC address yang dicatat oleh filter *blacklist* tidak dapat mengakses jaringan tersebut maupun dengan skema *whitelist*, dimana hanya perangkat dengan MAC yang dicatat oleh filter *whitelist* yang diperbolehkan mengakses jaringan tersebut.

1.3 Batasan Masalah

Batasan-batasan masalah dari penelitian ini adalah :

1. Pengujian dilakukan dengan perangkat yang mendukung penggantian MAC Address.
2. Pengujian dilakukan dengan perangkat yang mendukung penggantian MAC Address acak pada perangkat yang mendukung *MAC Address randomization* seperti mikrokontroler ESP8266 atau perangkat lainnya dengan dukungan sistem operasi seperti Android 6.0 dan seterusnya, dan Windows 10 dan seterusnya, serta pada *virtual NIC* pada *virtual enviorment* seperti VirtualBox.
3. Variable yang diambil dari penelitian ini adalah waktu *ping* atau *latency*, data *traceroute*, serta hasil *request / response* pada *application layer* pada

port HTTP (TCP 80 dan TCP 1111 untuk Android), serta beberapa data yang diambil dengan perangkat lunak penganalisa jaringan Wireshark

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah mengetahui dampak yang ditimbulkan pada jaringan apabila terjadi kesamaan MAC Address atau disebut juga dengan *MAC Address collision*.

1.5 Manfaat Penelitian

Manfaat penelitian ini adalah mengatasi kejadian yang tidak diinginkan apabila *MAC Address collision* memberikan dampak buruk atau gangguan dari suatu jaringan.

1.6 Metodologi Peneliatian

Metodologi penelitian yang telah dilakukan adalah sebagai berikut:

1. Menentukan topik yang akan dipilih dengan cara mengidentifikasi masalah-masalah yang ada saat ini untuk kemudian dicari solusinya. Pemilihan topik ini juga disertai konsultasi dengan dosen pembimbing.
2. Mengidentifikasi permasalahan-permasalahan yang ada, sehingga dapat ditemukan solusi dari masalah-masalah tersebut.
3. Pengumpulan data dilakukan dengan pengkajian dan pembelajaran lebih lanjut terhadap sistem yang akan dibuat, yaitu dengan cara:
    a. Studi literatur, yaitu mempelajari artikel, makalah, jurnal, karya tulis, situs web di internet, serta buku-buku yang terkait dengan *MAC Address collision*, *MAC Address spoofing*, dan *MAC Address randomization*.
    b. Konsultasi dengan dosen pembimbing dan dosen lain yang sesuai bidangnya mengenai rancangan sistem dan inovasi-inovasi yang akan diterapkan.

4. Membuat perancangan sistem, dimana MAC Address dari suatu perangkat dapat diganti, diacak, maupun disamakan dalam perangkat lain pada jaringan yang sama.
5. Penerapan rancangan yang telah dibuat untuk dilakukan sehingga menghasilkan data-data untuk dianalisis.
6. Data-data yang diperoleh dianalisis berdasarkan tujuan penelitian, sehingga dapat ditarik kesimpulan dari penelitian yang telah dilakukan.

1.7 Sistematika Penelitian

Usulan penelitian skripsi ini ditulis dengan susunan sebagai berikut:

BAB I: PENDAHULUAN

Bab ini berisi tentang latar belakang penelitian, batasan-batasan penelitian, tujuan dan manfaat penelitian, metodologi penelitian, serta sistematika penulisan usulan penelitian.

BAB II: TINJAUAN PUSTAKA

Bab ini berisi tentang penelitian-penelitian terdahulu mengenai penggunaan fitur *MAC Address Randomization,* hingga *MAC Address Spoofing*.

BAB III: LANDASAN TEORI

Bab ini berisi tentang teori-teori dasar yang mendukung penelitian ini, yaitu standar dari IEEE 802, hingga lapisannya yakni *Data Link Layer* dan *Physical Layer*.

BAB IV: METODE PENELITIAN

Bab ini berisi tentang penjelasan mengenai tahapan-tahapan penelitian secara keseluruhan yang meliputi bahan dan perangkat pada penelitian, rancangan arsitektur sistem, rancangan pelatihan dan pembelajaran, rancangan perangkat lunak, dan rancangan pengujian sistem.

BAB V: JADWAL PENELITIAN

Pada bab ini ditampilkan tabel tentang rencana waktu penelitian, dimulai dari penyusunan proposal hingga pembuatan laporan akhir.

BAB II

TINJAUAN PUSTAKA

Menrut penelitian Cardenas, Edgar D. "MAC Spoofing--An Introduction". GIAC Security Essentials Certification. SANS Institute 8 Februari 2013 berupa pemalsuan alamat MAC, hasilnya adalah penguji dapat memetakan arah lalu lintas internet ke tujuan yang bukan seharusnya, namun belum diuji apa yang terjadi jika alamat MAC sama dalam suatu jaringan, terutama sama dengan router.

Meskipun sejatinya alamat MAC tidak dapat diubah karena sudah ditentukan oleh pabrik pembuat *Network Interface Card (NIC)*, tetapi beberapa driver atau sistem operasi memperbolehkan untuk mengubah MAC address secara acak untuk alasan proteksi dari pelacakan pihak ketiga dan privasi, mengingat tiga buah set dari sepasang alamat heksadesimal merupakan *Organizationally Unique Identifier (OUI)* dapat diketahui perusahaan yang membuat perangkat NIC tersebut. (Android Open Source Project changes, 2015.).

Android Open Source Project merilis dokumentasi bahwa pada sistem operasi Android 10, *MAC Address Randomization* diaktifkan secara bawaan pada mode *client*, *SoftAP*, dan *Wi-Fi Direct* (Android Open Source Project changes, 2022.).

Berdasarkan penelitian dan catatan pembaharuan sistem operasi tersebut, diusulkan penelitian mengenai pengaruh dari *MAC Address Collision* dengan perangkat – perangkat yang mendukung perubahan MAC Address baik secara manual maupun diacak pada Tabel 2.1

Tabel 2.1 Tinjauan Pustaka

No	Peniliti	Judul	Keterangan
1	(Cardenas, Edgar D., 2013.)	"MAC Spoofing--An Introduction"	*MAC Address Spoofing*, pemetaan lalu lintas internet ke tujuan yang tidak semestinya.

2	(Android Open Source Project, 2015.)	Android 6.0 Changes	Dukungan pertama untuk mengacak *MAC Address* pada sistem operasi Android.
3	(Android Open Source Project, 2022.)	Implementing MAC Randomization	Dukungan penuh untuk mengacak *MAC Address* untuk alasan privasi, dimana *MAC Address randomization* diaktifkan secara *default*.
4	(VirtualBox documentation)	Oracle VM VirtualBox User Manual	Dukungan untuk mengganti *MAC Address* untuk perangkat *virtual network adapter*.

BAB III

LANDASAN TEORI

3.1 Infrastruktur jaringan

Jaringan lokal atau *Local Area Network* (LAN) merupakan jaringan komputer yang terbatas pada suatu area kecil, seperti pada jaringan rumah, gedung perkantoran maupun kampus. Perangkat pada jaringan lokal pada umumnya diberi alamat IP privat *bogon* oleh *server Dynamic Host Configuration Protocol* (DHCP). IP *bogon* adalah IP yang tidak dialokasikan dan tidak boleh digunakan menurut standar *Internet Assigned Numbers Authority* (IANA) yang memiliki rentang yang juga telah distandarisasi dan tidak dapat diakses secara langsung dari internet.

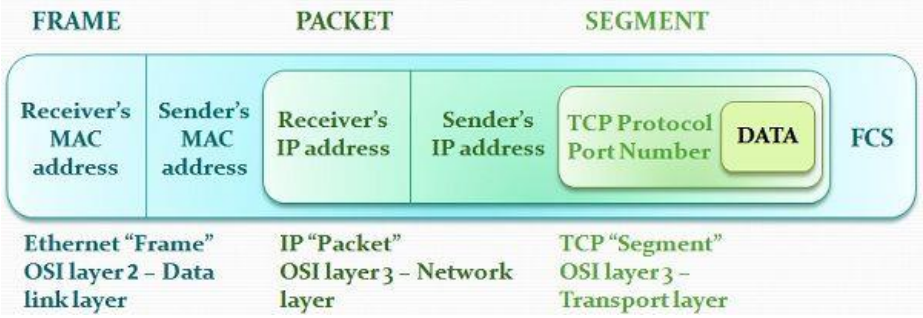
Beberapa contoh rentang alamat IP *bogon* adalah 10.0.0.0/8. Artinya, dalam jaringan ini, jumlah perangkat yang dapat terhubung adalah 16.777.214 dengan rentang IP 10.0.0.1 sampai dengan 10.255.255.254. Contoh rentang alamat IP *bogon* berikutnya adalah 172.16.0.0/12, yang dapat memuat 1.048.574 buah perangkat dengan rentang IP 172.16.0.1 sampai 172.16.255.254. Contoh rentang alamat lainnya adalah 192.168.1.0/24, yang dapat memuat 254 buah perangkat yang dengan rentang IP 192.168.1.1 sampai dengan 192.168.1.254.

Untuk mempermudah pemberian IP *bogon* tersebut, digunakan server DHCP pada *router* atau *gateway*. *Dynamic Host Configuration Protocol* (DHCP) adalah protokol jaringan yang digunakan untuk memberi IP dalam suatu jaringan privat, serta konfigurasi – konfigurasi lainnya seperti *subnet mask*, IP *gateway* dan IP server DNS, kepada perangkat yang terhubung pada suatu jaringan.

Pada penelitian ini, akan dibuat suatu model jaringan lokal yang tersusun atas perangkat yang berjalan pada IP *bogon*, dengan bantuan server DHCP, baik pada lingkungan fisik maupun lingkungan virtual pada VirtualBox.

3.2 MAC Address

Skema pengalamatan MAC Address *Institute of Electrical and Electronics Engineers* (IEEE) 802 yang digunakan di dalam jaringan Ethernet, Wi-Fi, maupun Bluetooth berasal dari Xerox Network Systems yang dikembangkan oleh perusahaan asal Amerika Serikat yang bernama Xerox pada tahun 1970-an, kemudian ditetapkan oleh IEEE pada tahun 1980 dengan membentuk komite penetapan standar teknologi jaringan komputer, yang memberikan pengaruh terhadap perkembangan model jaringan *Open System Interconnection (OSI)* yang juga mempengaruhi perkembangan jaringan *Local Area Network (LAN)*. Skema pengalamatan MAC ditetapkan dan diadopsi oleh standar IEEE 802.3 dan telah menjadi dasar oleh berbagai teknologi jaringan. *Organizationally Unique Identifier* (OUI) dengan kode 00:00:00, 00:00:03 sampai 00:00:08 merupakan kode OUI dari Xerox itu sendiri. Dengan skema pengalamatan sebesar 48-bit, terdapat 281 triliun kombinasi alamat MAC yang berbeda.



Gambar 3.1 sumber :

https://techdifferences.com/wp-content/uploads/2017/08/featured-4.jpg

Pada *Ethernet frame (OSI Layer 2)*, terdapat *MAC Address* penerima, *MAC Address* pengirim, baru kemudian IP penerima dan IP pengirim *(OSI Layer 3)*. Keberadaan *OSI Layer 2* dimana MAC Address memainkan perannya menjadi

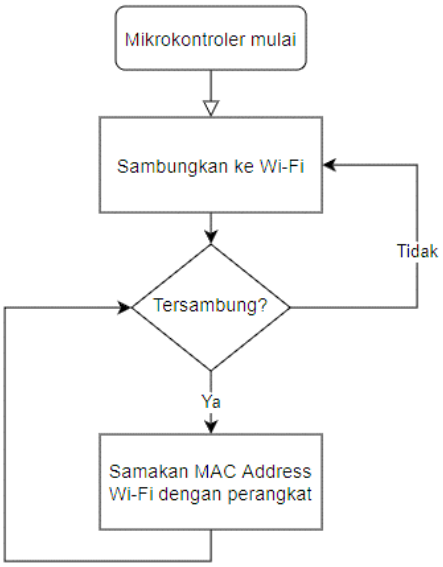
peranan penting setelah *physical layer* karena berada di lapisan bawah dari *network layer* dan menjadi cara perangkat lain untuk mengenali identitas fisiknya yang bersifat unik.

Seiring perkembangannya, *driver, software,* dan sistem operasi terbaru dari NIC memperbolehkan untuk merubah *MAC Address* untuk mencegah *tracking* dari pihak ketiga. Di sisi lain, sifat *hard-coded* dari *MAC Address* tidak berlaku lagi. Kemudahan mengganti *MAC Address* memiliki dampak baik bagi pengguna karena *MAC Address* yang diacak *(randomized)* menyulitkan pihak ketiga untuk melakukan pelacakan, sehingga pengguna yang menggunakan fitur *MAC Address Randomization* mendapatkan rasa aman yang lebih. Namun, cara ini juga dapat digunakan untuk mem-*bypass* fitur *MAC Address filter* pada *router*, sehingga metode ini akan menjadi tidak efektif jika semakin banyak perangkat menggunakan fitur *MAC Address randomization* yang diterapkan secara *default*. Selain itu ada metode pemalsuan *MAC Address* yang umum disebut sebagai *MAC Address spoofing,* dimana pihak yang tidak bertanggungjawab mengganti *MAC Address* dari perangkatnya untuk melakukan sabotase pada jaringan yang dituju.

Selain *randomization* dan *spoofing,* terdapat juga kejadian dimana dua atau lebih perangkat memiliki *MAC Address* yang sama, yang disebut juga dengan *MAC Address collision*. Pada penelitian ini, akan dilakukan percobaan untuk kejadian ini, dimana beberapa perangkat akan disamakan *MAC Address* nya.

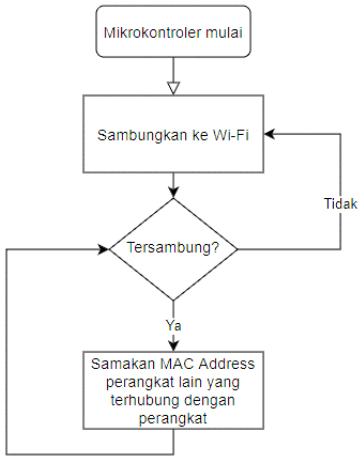
3.3 Penggantian MAC Address

Dengan mikrokontroler sederhana berbasis Wi-Fi yang MAC Address nya dapat diganti, pengujian pengaruh MAC Collision dan MAC Spoofing dapat dilakukan. Perangkat yang dipilih pada penelitian ini adalah NodeMCU dan Android yang telah di root sehingga MAC Address nya dapat diubah.



Gambar 3.2 Konfigurasi Pertama

Dengan konfigurasi berikut, perangkat penguji akan mengambil MAC Address dari router yang dituju. Sebelum tersambung, *MAC Address* perangkat penguji adalah *default* atau diacak terlebih dahulu, kemudian setelah tersambung dilakukan *scanning* untuk mendapatkan *MAC Address* dari router.



Gambar 3.3 Konfigurasi Kedua

Dengan konfigurasi berikut, perangkat penguji akan mengambil *MAC Address* dari perangkat lain yang terhubung. Sebelum tersambung, *MAC Address* perangkat penguji adalah *default* atau diacak terlebih dahulu, kemudian setelah tersambung dilakukan *scanning* untuk mendapatkan *MAC Address* dari perangkat lain dalam jaringan yang sama.

3.4 Variabel terikat yang akan digunkan

Variabel terikat yang akan diteliti adalah *latency*, data *traceroute*, dan data hasil pengamatan Wireshark.

Latency merupakan waktu yang dibutuhkan data untuk melakukan perjalanan dari satu titik ke titik lain dalam suatu jaringan. *Latency* diukur dengan cara menghitung waktu yang telah dilalui pada saat mengirim data dari sumber hingga menerima respon dari tujuan, yang umumnya dinyatakan dalam satuan milidetik (ms) atau mikrodetik (µs). *Latency* umumnya dipengaruhi oleh beberapa faktor, seperti jarak fisik antara sumber dan tujuan, jenis teknologi yang digunakan, dan jumlah perangkat yang terhubung dalam suatu jaringan.

Umumnya untuk melakukan pemeriksaan *latency*, digunakan utilitas yang bernama *ping*. *Ping* adalah suatu perintah yang tersedia di hampir semua sistem operasi (OS) yang umum digunakan, seperti Windows, macOS, Linux, dan UNIX. *Ping* berjalan pada protokol Internet Control Message Protocol (ICMP) untuk mengirim dan menerima respons dari perangkat yang dituju. Selain untuk mengukur *latency*, *ping* digunakan untuk memeriksa konektivitas antar perangkat dalam suatu jaringan. *Ping* berjalan pada layer OSI ke-3 atau *Network Layer*. Dengan demikian, variabel waktu *ping* atau *latency* digunakan dalam penelitian ini sebagai variabel terikat.

Selanjutnya, variabel terikat yang akan diuji berikutnya adalah data *traceroute*. *Traceroute* adalah utilitas yang ada juga di OS yang umum digunakan

yang digunakan untuk melacak jalur yang dilalui paket data yang dikirim dari sumber ke tujuan melalui *router*. *Traceroute* menggunakan protokol ICMP dan *User Datagram Protocol* (UDP) dalam pengoperasiannya. ICMP pada *traceroute* mengirimkan paket dengan tipe pesan *echo request* atau *ping* ke tujuan dan *traceroute* menganalisis respons yang diterima dari setiap *router* atau *hop* yang dilaluinya untuk menunggu balasan pesan ICMP *echo reply* atau *pong*. Data yang diberikan oleh utilitas *traceroute* berupa alamat IP dan waktu respon dari setiap *router* dan banyaknya *hop* yang dilalui. UDP pada *traceroute* mengirimkan paket UDP ke *port* tertentu menuju perangkat tujuan dan kemudian *traceroute* menganalisis respons dari setiap *router* yang dilewati oleh paket. *Traceroute* mampu mengetahui informasi alamat IP dari setiap *hop* dalam jalur dengan teknik *Time to Live* (TTL), yaitu nilai dalam paket data yang akan digunakan dalam *traceroute* untuk menentukan batasan waktu yang diperbolehkan untuk paket data serta menentukan jumlah hop maksimum yang boleh dilewati sebelum dihapus oleh *router*. Nilai TTL bertambah satu untuk setiap *hop* yang dilalui, dimana pada penelitian ini hanya digunakan satu buah *router* atau *hop* saja (baik dalam pegujian fisik maupun secara virtual) yang dilalui dalam keadaan ideal. Dengan demikian, data *output* dari perangkat lunak *traceroute* digunakan sebagai variabel terikat dalam penelitian ini.

Variabel penelitian selanjutnya adalah HTTP *request*. *Hypertext Transfer Protocol* (HTTP) adalah protokol aplikasi yang berjalan pada lapisan OSI ke-7, sekaligus lapisan tertinggi dari lapisan OSI atau disebut juga *application layer*. HTTP berjalan pada *application layer* karena HTTP merupakan protokol untuk aplikasi yang umum digunakan, khususnya aplikasi web. HTTP merupakan protokol *client-server*, dimana keduanya merupakan perangkat lunak yang berkomunikasi satu sama lain dengan cara *client* melakukan permintaan atau *request* terlebih dahulu ke *server* melalui HTTP *request*. HTTP *request* adalah pesan yang dikirim oleh *client* ke *server* untuk meminta sumber daya yang diinginkan. HTTP *request*, yang berjalan pada *application layer* ini menggunakan tiga buah informasi dasar, yaitu metode atau *method*, *Uniform Resource Identifier*

(URI) atau *Uniform Resource Locator* (URL), dan versi protokol HTTP yang akan digunakan *client*. Metode HTTP yang akan digunakan dalam penelitian ini adalah metode GET, yang merupakan metode *default*, baik dalam *browser* atau perangkat lunak cURL, sedangkan URI atau URL yang akan digunakan adalah alamat IP dari perangkat – perangkat penelitian, baik dalam lingkungan fisik maupun lingkungan percobaan virtual. Dengan demikian, data *output* dari permintaan HTTP GET digunakan sebagai variabel terikat dalam penelitian ini.

Variabel terikat penelitian yang terakhir adalah data hasil pengamatan oleh Wireshark. Wireshark merupakan perangkat lunak untuk melakukan analisis jaringan dengan cara memantau lalu lintas jaringan dan menampilkan paket data yang dikirim dan diterima pada suatu jaringan. Wireshark mampu menganalisis paket data, termasuk MAC tujuan dan MAC sumber, serta akan digunakan pada penelitian berikut, terutama pada percobaan lingkungan virtual dalam VirtualBox.

BAB IV

METODE PENELITIAN

Bab ini membahas langkah-langkah yang dilakukan pada penelitian skripsi. Beberapa alat dan bahan, tahapan, rancangan, serta pengujian akan dijelaskan selanjutnya.

4.1 Alat dan Bahan

Penelitian ini menggunakan perangkat yang tertera pada tabel 4.1 untuk pengujian *MAC Address Collision*. Peralatan pada tabel 4.2 merupakan perangkat lunak penunjang.

Tabel 4.1 Daftar Komponen Penunjang

No.	Nama Komponen	Fungsi
1.	NodeMCU dengan mikrokontroler ESP8266	Perangkat *client* yang MAC Address nya akan diubah.
2.	Ponsel dengan sistem operasi Android *(rooted).*	Perangkat *client* yang MAC Address nya akan diubah.
3.	*Router* yang akan diuji	Perangkat *router* yang MAC Address nya akan disamakan.

Tabel 4.2 Daftar Peralatan Lunak Penunjang

No.	Nama Alat	Fungsi
1.	Arduino IDE	Untuk perubahan MAC Address.
2.	MAC Address changer (Android)	Untuk perubahan MAC Address.
3.	VirtualBox	Untuk percobaan lanjutan pada mesin vitual.
4.	Wireshark	Untuk menganalisis paket data
5.	cURL atau browser	Untuk melakukan HTTP GET *request*

6.	Traceroute	Untuk menampilkan TTL dan *hop* yang terlibat

Perangkat keras dan perangkat lunak dipastikan dapat diubah *MAC Address* nya. Percobaan juga dilakukan dalam mesin virtual pada program VirtualBox, dimana *virtual NIC* dapat dilakukan penggantian *MAC Address*.

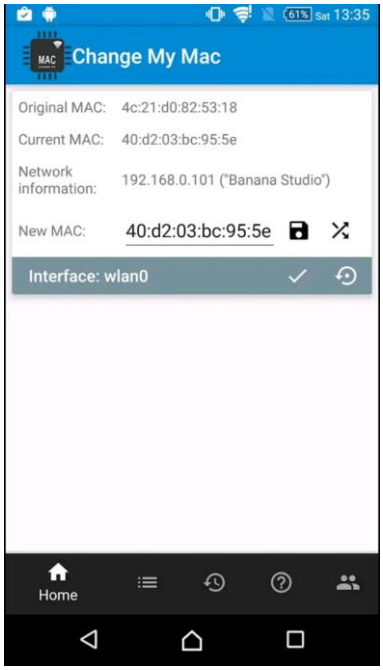
Percobaan pada hasil pra-proposal dilakukan pada router dan AP ZTE F609 dengan versi perangkat lunak V7.0.10P1N14. Perangkat keras yang digunakan pada percobaan ini adalah NodeMCU dengan mikrokontroler ESP8266 yang memiliki spesifikasi CPU 32-bit RISC yang berjalan pada kecepatan 80MHz, memori sebear 80kB, penyimpanan flash sebesar 4MB, serta mampu berjalan pada protokol Wi-Fi 2.4GHz 802.11/b/g/n. Dukungan untuk mengganti alamat MAC menjadi alasan mengapa mikrokontroler ini digunakan pada penelitian ini. Penggantian alamat MAC pada ESP8266 dapat dilakukan dengan *software development kit* (SDK) bawaan atau dengan Arduino ESP8266 Core. Dalam penelitian ini, digunakan Arduino IDE dengan tambahan pustaka ESP8266WiFi. Mengganti alamat MAC pada Arduino IDE dilakukan dengan fungsi dari *wifi_set_macaddr(STATION_IF, new_mac)*.

```
1  #include <ESP8266WiFi.h>
2
3  void setup() {
4    uint8_t new_mac[6] = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
5    wifi_set_macaddr(STATION_IF, new_mac);
6  }
7
8  void loop() {
9
10 }
11
```

Gambar 4.1 Penggantian alamat MAC pada ESP8266

Penggantian alamat MAC NodeMCU dilakukan sesuai dengan skenario penelitian yang akan diuji.

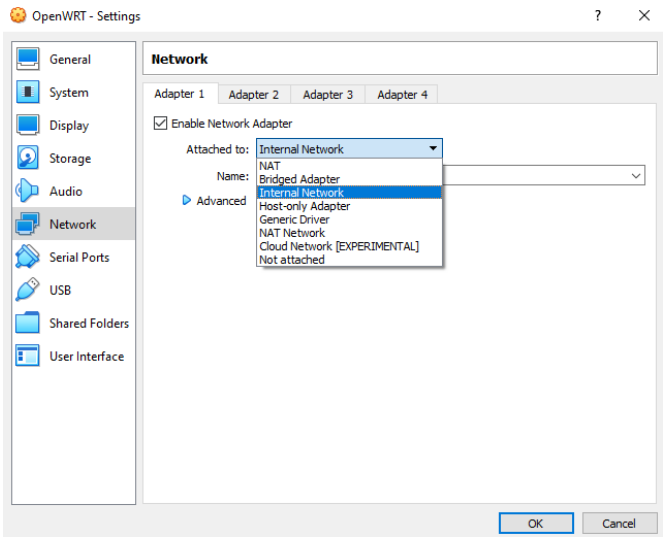
Penggantian alamat MAC pada perangkat Android dapat dilakukan pada Android yang telah di-*root* atau memiliki akses *root*. Dengan diberikannya akses *root*, maka aplikasi pihak ketiga mampu mengganti alamat MAC sesuai dengan skenario percobaan. Perangkat yang digunakan adalah Xiaomi Redmi Note 5, dengan alamat MAC 80:35:C1:3E:3B:30. 80:35:C1 merupakan tiga oktet pertama yang merupakan *Organizationally Unique Identifier* (OUI) dari Xiaomi Communications Co Ltd. Proses penggantian MAC pada Android dapat dilakukan dengan memasang aplikasi pihak ketiga yang mampu mengubah alamat MAC dengan dukungan akses *root*. Pada penelitian ini, digunakan aplikasi "Change My MAC – Spoof Wifi MAC" yang dikembangkan oleh Banana Studio. Selain menggunakan aplikasi pihak ketiga, mengganti alamat MAC juga dapat dilakukan dengan memodifikasi file konfigurasi wpa_supplicant yang merupakan utilitas *system-level*.



Gambar 4.2 Penggantian alamat MAC pada Android

Penggantian alamat MAC pada VirtualBox dapat dilakukan dengan membuat NIC atau *network adapter* pada beberapa mesin virtual, atau *virtual machine* (VM), terhubung pada *internal network*.

Dalam VirtualBox, *internal network* adalah tipe jaringan yang memungkinkan mesin virtual (VM) berkomunikasi satu sama lain tanpa perlu terhubung atau terlihat dari luar *virtualized environment* itu sendiri. *Internal network* setara dengan *ethernet switch* dalam wujud fisiknya, sehingga simulasi jaringan yang melibatkan beberapa VM ini dapat berkomunikasi satu sama lain tanpa perlu terhubung ke *Network Address Translation* (NAT). NAT pada VirtualBox, sekaligus opsi *default* dari *"Attached to:"* adalah adaptor jaringan yang memungkinkan VM terhubung ke jaringan lokal dari *host* atau komputer yang menjalankan VirtualBox tersebut. Mode NAT pada VirtualBox bekerja layaknya sebuah *gateway* atau *router*, dimana *gateway* nya adalah komputer *host* yang menjalankan VirtualBox. NAT pada VirtualBox juga memiliki server *Dynamic Host Configuration Protocol* (DHCP) yang mampu memberikan alamat IP ke beberapa VM yang terhubung secara otomatis.

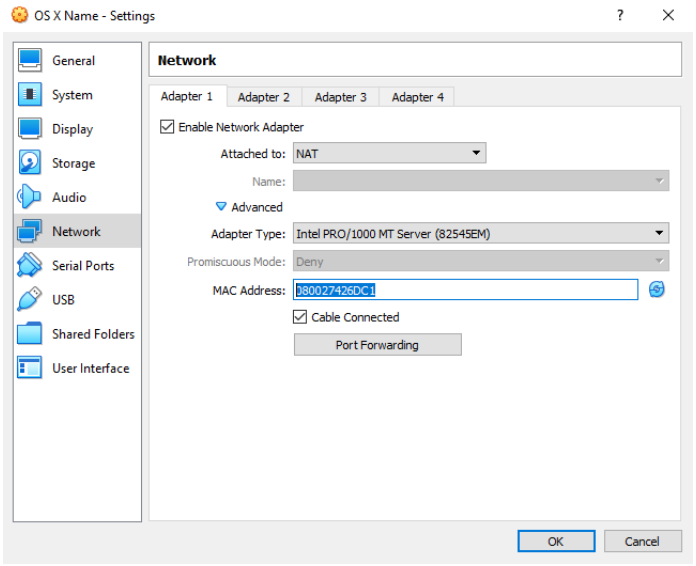


Gambar 4.3 Internal Network

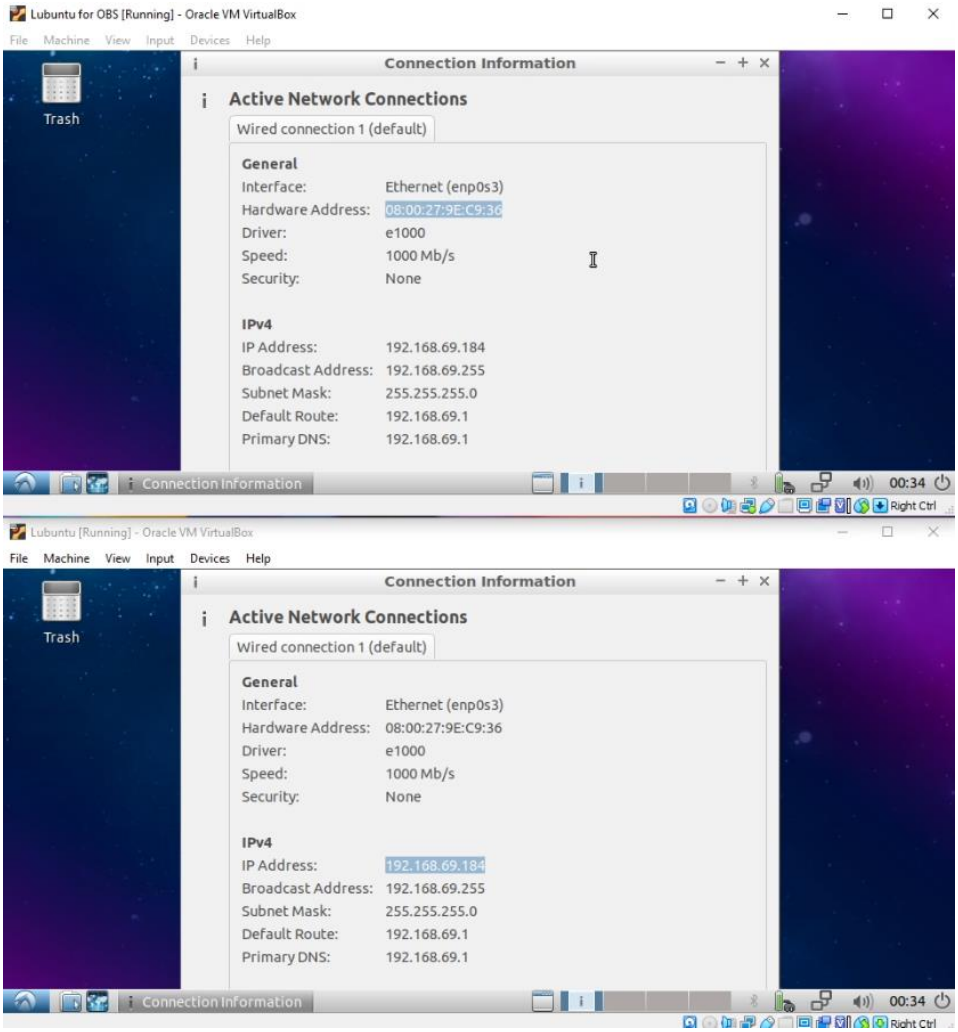
Tidak seperti mode NAT, dimana terdapat server DHCP, *Internal network* hanya setara selayaknya *ethernet switch* dan tidak memiliki fitur DHCP, sehingga

setiap VM harus dikonfigurasikan alamat IP yang unik untuk setiap VM, IP *default gateway*, *subnet mask*, dan server *Domain Name System* (DNS) secara manual. Namun, karena pada penelitian ini akan digunakan sebuah VM yang bertindak sebagai *gateway* dan *router* OpenWRT, maka server DHCP akan sekaligus dikonfigurasikan pada VM OpenWRT tersebut.

Untuk kebutuhan penelitian, dibutuhkan juga NIC yang mampu diubah alamat MAC nya. VirtualBox memiliki dukungan untuk mengubah alamat MAC langsung pada *network adapter* pada setiap VM. Pada dasarnya, VirtualBox memberikan alamat MAC yang unik untuk setiap NIC, namun penggantian alamat MAC secara manual dapat juga dilakukan.

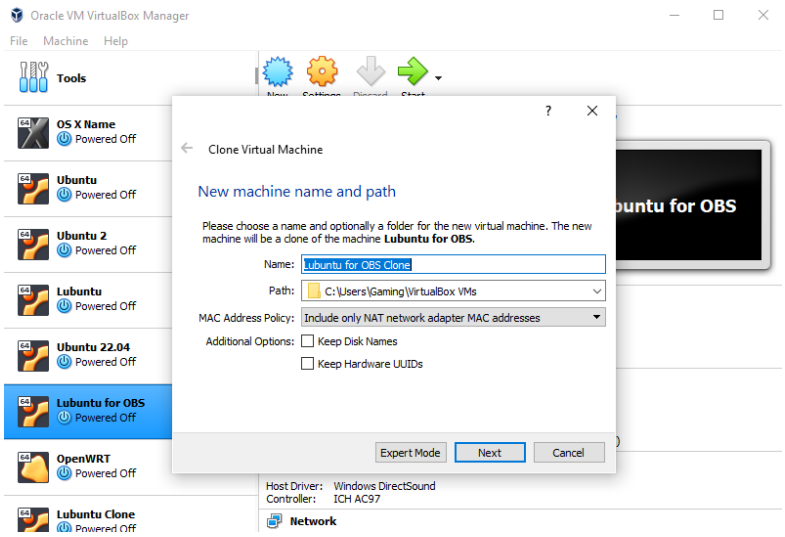


Gambar 4.4 Penggantian MAC Address pada VirtualBox



Gambar 4.5 Dua buah VM dengan MAC address yang sama

Untuk menduplikat VM yang identik, sebagai batasan variabel kontrol, VM VirtualBox dapat di-*clone* dengan fitur *Clone Virtual Machine*. Dengan demikian, terbentuk dua buah VM identik dengan sistem operasi dan *disk* yang identik.



Gambar 4.6 Clone Virtual Machine

Percobaan dengan *ping* dilakukan dengan menjalankan perintah *ping* di berbagai OS yang mendukungnya. Pada penelitian ini, semua OS mendukung operasi *ping*. *Ping* dilakukan dengan sintaks *ping*, kemudian diikuti dengan alamat IP atau *hostname*.

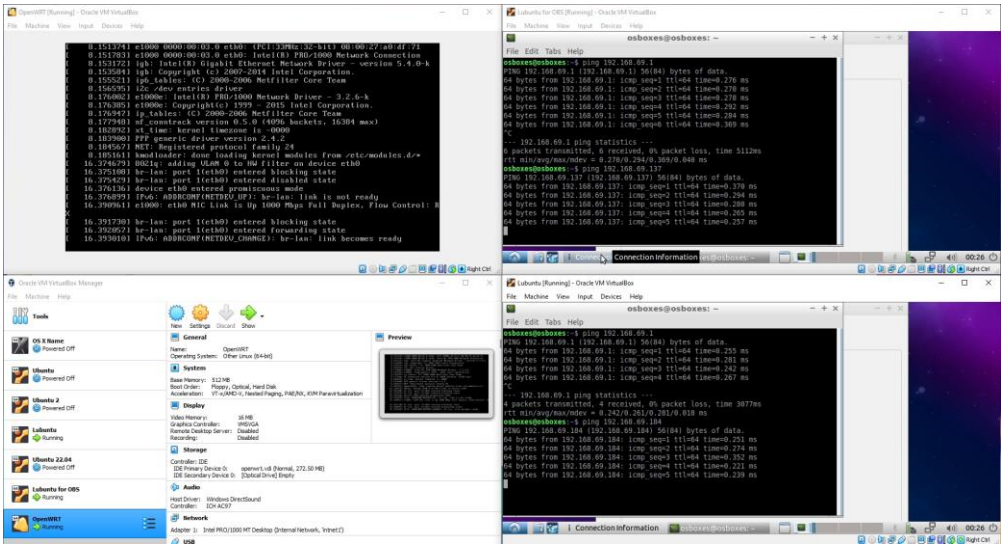
```
C:\Users\Gaming>ping gabrielkheisa.xyz

Pinging gabrielkheisa.xyz [185.199.108.153] with 32 bytes of data:
Reply from 185.199.108.153: bytes=32 time=24ms TTL=57
Reply from 185.199.108.153: bytes=32 time=25ms TTL=57
Reply from 185.199.108.153: bytes=32 time=24ms TTL=57
Reply from 185.199.108.153: bytes=32 time=25ms TTL=57

Ping statistics for 185.199.108.153:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 24ms, Maximum = 25ms, Average = 24ms

C:\Users\Gaming>_
```

Gambar 4.7 Ping pada sistem operasi Windows 10



Gambar 4.8 Ping pada dua buah VM dengan sistem operasi Linux Lubuntu pada VirtualBox menuju VM OpenWRT

Variasi ping rata – rata atau *average* akan digunakan sebagai variabel terikat dalam penelitian ini. Pengujian *ping* akan dilakukan pada lingkungan fisik dan lingkungan virtual pada VirtualBox.

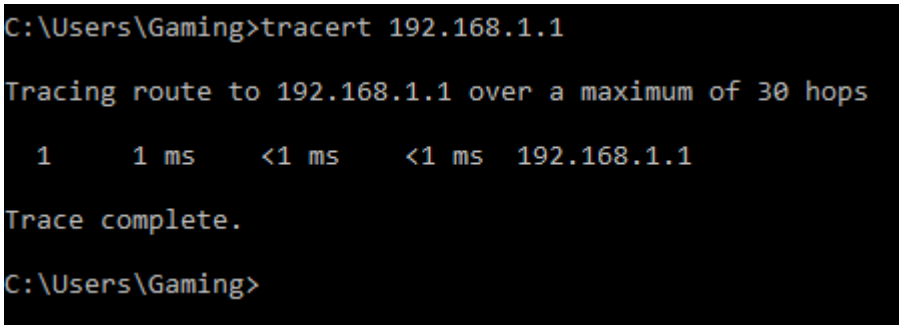
Percobaan dengan *traceroute* dilakukan dengan menjalankan perintah *traceroute* pada berbagai OS yang mendukungnya. Pada Windows, *traceroute* dapat dilakukan dengan cara mengetik *tracert* pada *command prompt*, diikuti dengan hostname atau IP yang ingin dituju.

```
C:\Users\Gaming>tracert ugm.ac.id

Tracing route to ugm.ac.id [175.111.88.3]
over a maximum of 30 hops:

  1     1 ms    <1 ms    <1 ms  192.168.1.1
  2     *        *     1346 ms  36.73.112.1
  3     2 ms     2 ms     2 ms  125.160.1.197
  4     4 ms     5 ms     2 ms  118.97.5.165
  5     3 ms     4 ms     2 ms  118.97.5.166
  6     2 ms     2 ms     3 ms  host-202-43-92-46.ugm.ac.id [202.43.92.46]
  7     4 ms     3 ms     4 ms  ugm.ac.id [175.111.88.3]
```

Gambar 4.9 Traceroute menuju ugm.ac.id

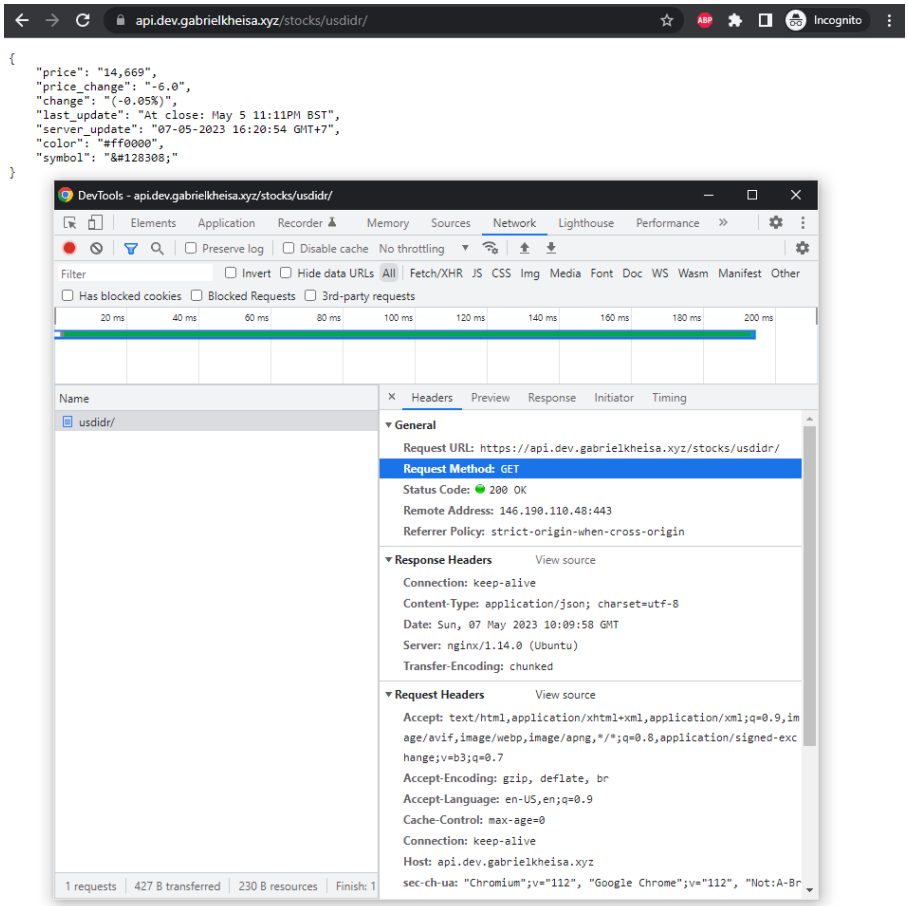


Gambar 4.10 Traceroute menuju gateway yang hanya melewati satu buah hop

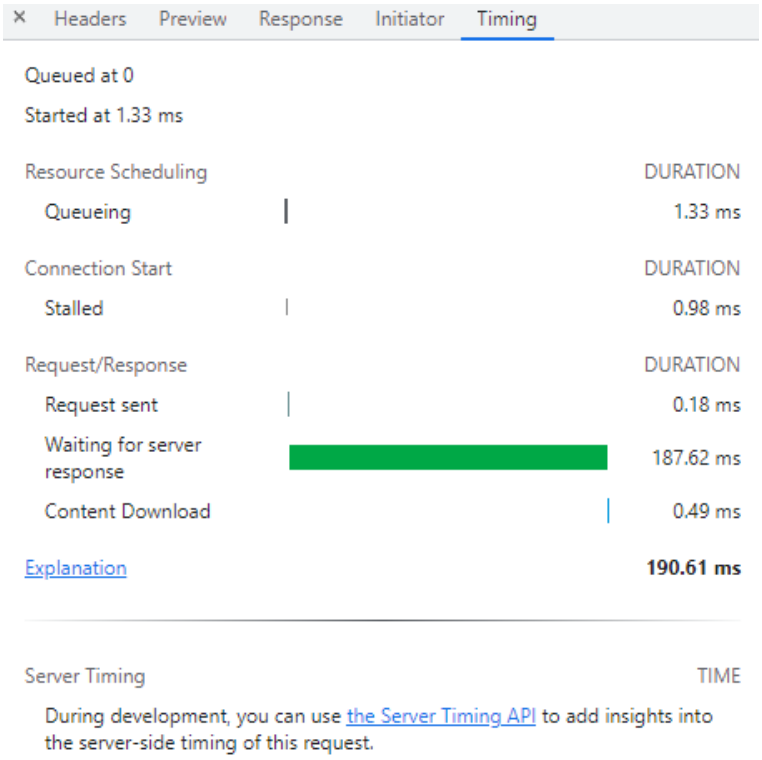
Percobaan dengan *traceroute* dilakukan dengan menjalankan perintah *traceroute* pada berbagai OS yang mendukungnya. Pada Windows, *traceroute* dapat dilakukan dengan cara mengetik “tracert” pada *command prompt*, diikuti dengan hostname atau IP yang ingin dituju.

Meskipun *traceroute* hanya merupakan utilitas untuk melakukan pelacakan rute paket, data hasil *traceroute* digunakan sebagai variabel terikat pembanding antara kondisi jaringan dalam keadaan ideal dengan kondisi jaringan dalam keadaan dimana terjadi MAC *address collision*.

Selanjutnya adalah menganalisa pengaruh MAC *address coliision* dengan menggunakan HTTP GET atau cURL untuk mengamati pengaruh MAC *address collision* pada *application layer* pada model lapisan OSI. HTTP GET dapat dilakukan melalui *browser* pada umumnya dengan memasukkan alamat IP atau URL ke *address bar*. Hal ini setara dengan HTTP GET yang dilakukan oleh cURL GET.

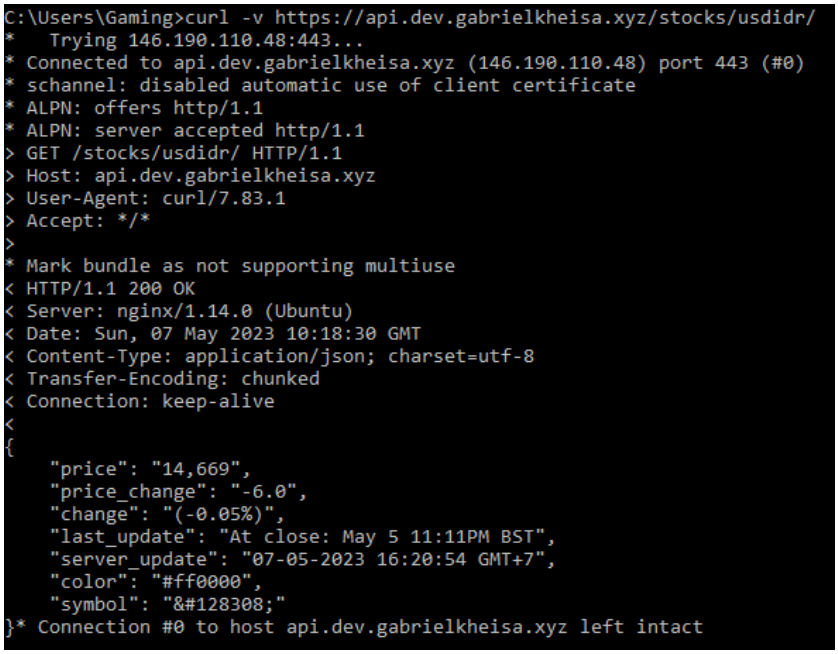


Gambar 4.11 HTTP GET pada browser Google Chrome pada developer options, memasukkan alamat URL atau IP ke adress bar akan membuat browser mengirimkan permintaan HTTP GET ke server dengan alamat yang dituju



Gambar 4.12 Informasi timing untuk mengamati waterfall waktu yang dibutuhkan

Pada Windows, cURL dapat dilakukan dengan cara mengetik “curl” pada *command prompt*, diikuti dengan *flag* “-v” untuk menampilkan statistik dan informasi tambahan pada saat cURL dijalankan, kemudian diikuti dengan hostname atau IP yang ingin dituju.



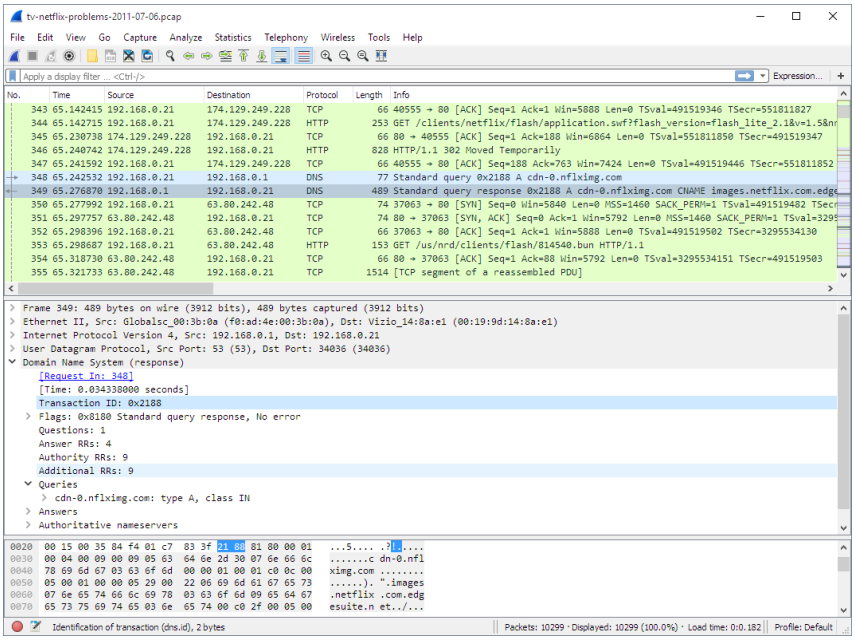
Gambar 4.13 HTTP GET pada perangkat lunak cURL pada sistem operasi Windows 10

Meskipun *traceroute* hanya merupakan utilitas untuk melakukan pelacakan rute paket, data hasil *traceroute* digunakan sebagai variabel terikat pembanding antara kondisi jaringan dalam keadaan ideal dengan kondisi jaringan dalam keadaan dimana terjadi MAC *address collision*.

Adanya respon dari HTTP GET, yang berupa *payload* teks, HTML, XML, JSON, atau jenis data yang lain akan dijadikan variabel terikat dalam penelitian berikut. Selain itu, data *waterfall* berupa waktu juga dijadikan variabel terikat dalam penelitian berikut.

Selanjutnya adalah pengamatan data hasil analisa Wireshark. Wireshark dapat meneliti data yang dikirim melalui jaringan serta protokol – protokol umum seperti protokol jaringan yang terdiri dari TCP, UDP, ICMP, IP, dan ARP, serta meneliti protokol aplikasi, seperti HTTP, FTP, SSH, SMTP, DNS, dan sebagainya. Karena pada variabel – variabel terikat sebelumnya yang menganalisa HTTP GET dan *ping*, Wireshark dapat menampilkan informasi yang lebih luas, seperti *collision*, *congestion*, dan paket data yang hilang atau rusak, serta statistik – statistik jaringan

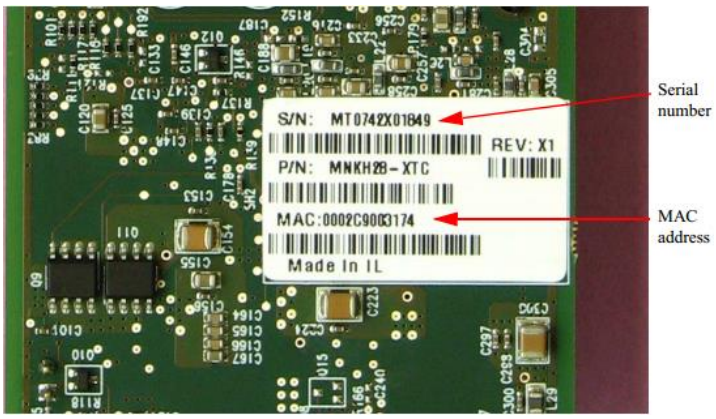
lainnya seperti jumlah paket data yang hilang, jumlah paket yang diterima, dan kegagalan transmisi.



Gambar 4.14 Wireshark yang berjalan pada Windows 10, sumber: https://www.wireshark.org/

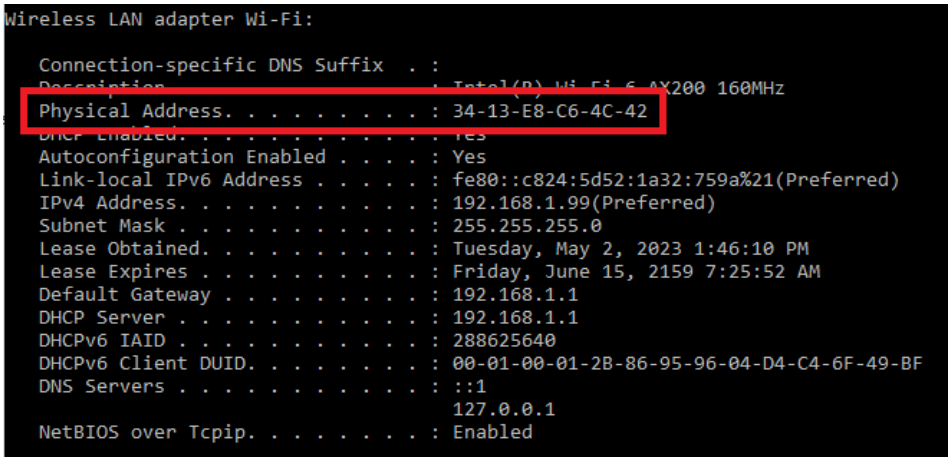
Untuk memperoleh alamat MAC yang akan dijadikan variabel bebas, dapat dilakukan dengan cara mengamati langsung alamat MAC yang tertera pada kartu fisik jaringan.

Gambar 4.15 Alamat MAC WiFi pada NIC Intel AX200, sumber: https://m.media-amazon.com/images/I/616Fc7j0HlL._AC_UF894,1000_QL80_.jpg

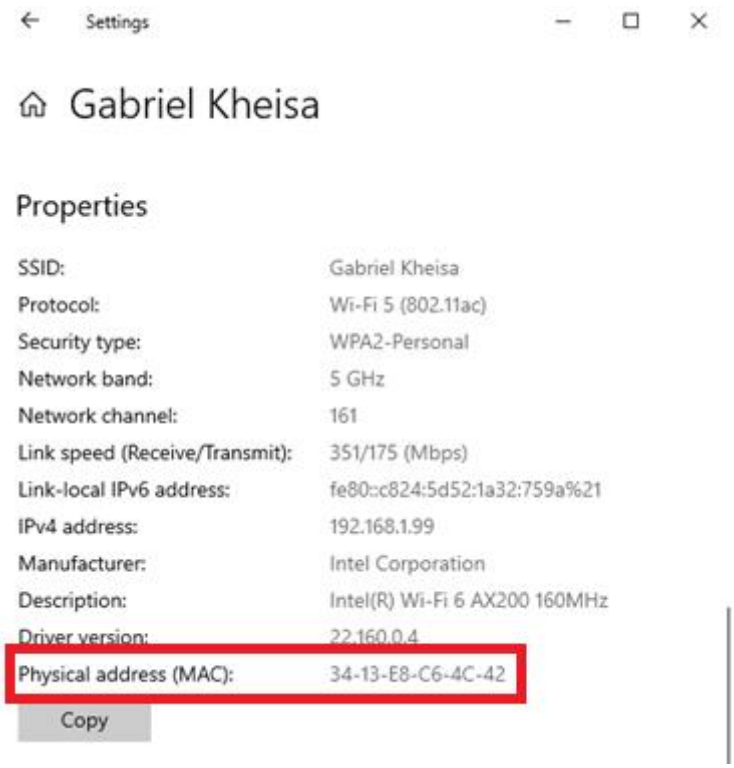


Gambar 4.16 Alamat MAC pada NIC Mellanox, sumber: https://docs.nvidia.com/networking/display/PreBootDriversv80/Case+II:+Ethernet+Ports

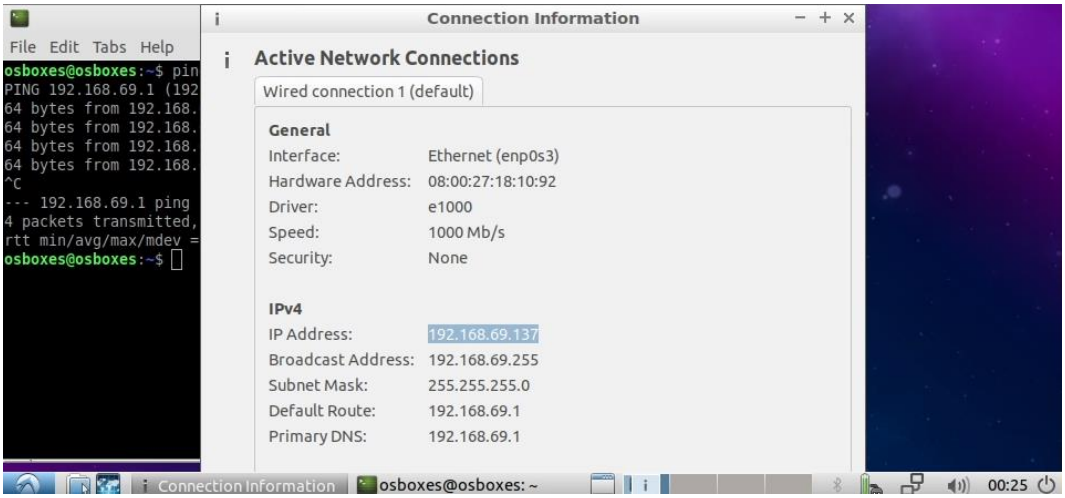
Namun, alamat MAC juga dapat diperoleh dengan utilitas yang ada pada sistem operasi Windows, seperti *ipconfig* pada *command prompt* dan *ifconfig* pada sistem operasi Linux atau pada pengaturan jaringan pada masing – masing sistem operasi, dan juga dapat diketahui oleh *administrator* jaringan yang memiliki akses ke *router*.



Gambar 4.17 Alamat MAC atau physical address pada command prompt Windows



Gambar 4.18 Alamat MAC atau physical address pada pengaturan pada sistem operasi Windows 10



Gambar 4.19 Alamat MAC atau physical address pada pengaturan pada sistem operasi Linux Lubuntu

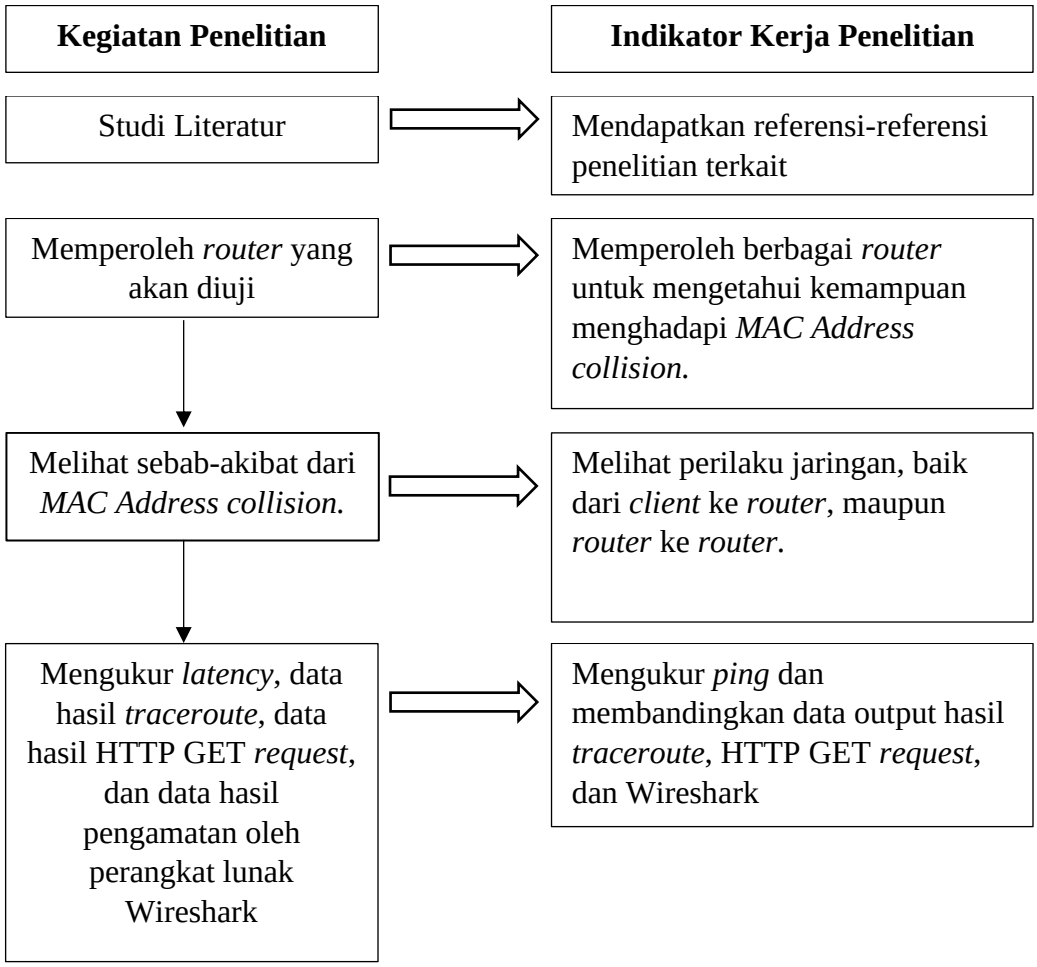
Allocated Address

MAC Address	IP Address	Remaining Lease Time	Host Name	Port
1c:83:41:1a:3a:07	192.168.1.102	infinity	7879c7c5	LAN3
1a:4a:01:a8:0b:78	192.168.1.6	32335		LAN2
c4:74:1e:b7:0b:ee	192.168.1.101	infinity		LAN2
34:13:e8:c6:4c:42	192.168.1.99	infinity	3AhVEPKWCZWld	LAN2
84:f3:eb:cc:03:ef	192.168.1.13	44528	ESP-CC03EF	LAN2
6a:aa:b8:65:5d:a6	192.168.1.10	68130	POCO-X3-Pro	LAN2
d4:a6:51:d9:31:b8	192.168.1.4	57981		LAN2
a4:30:7a:92:0f:8a	192.168.1.2	41656	Samsung	LAN2
b4:b0:24:df:ee:8d	192.168.1.5	59236	TL-WR840N	LAN2
b4:fb:e3:31:42:20	192.168.1.7	59244	MV45136856	LAN2
44:01:bb:5e:1a:91	192.168.1.8	59241	MV48667020	LAN2
82:a2:64:a9:8c:36	192.168.1.3	37014		LAN2
c4:dd:57:34:d5:29	192.168.1.9	78288	ESP_34D529	LAN2

Gambar 4.20 Alamat MAC atau physical address pada tabel DHCP dari router ZTE F609

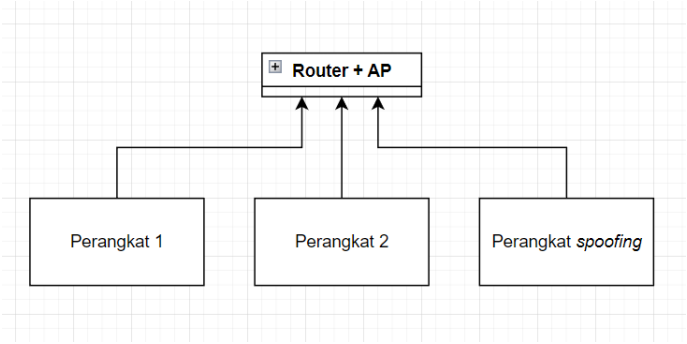
4.2 Tahapan Penelitian

Penelitian skripsi ini memiliki tahapan penelitian yang terbagi menjadi beberapa tahap sebagaimana skema pada Gambar 4.1.

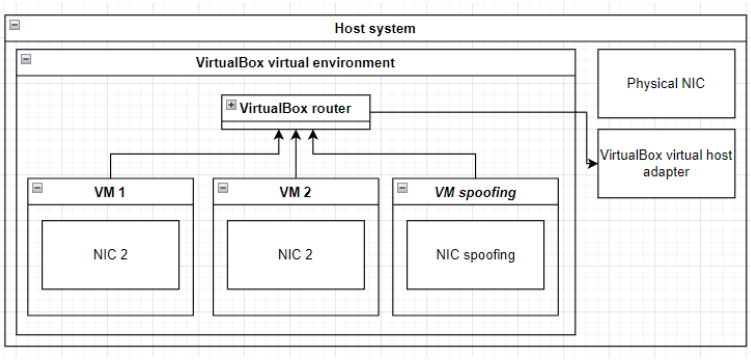


Gambar 4.6 Tahapan Penelitian

4.3 Rancangan sistem



Gambar 4.8 Topologi pertama (fisik)



Gambar 4.9 Topologi kedua (virtual environment)

4.4 Program NodeMCU untuk lingkungan fisik

```
1  #include <ESP8266WiFi.h>
2
3  #include <ESP8266Ping.h>
4
5  #include <ESP8266HTTPClient.h>
6
7  #include <WiFiUDP.h>
8
9  #include <ESP8266WebServer.h>
```

Gambar 4.10 Library yang digunakan pada NodeMCU sebagai variabel bebas

```
// 84:F3:EB:CC:03:EF default MAC
// uint8_t newMACAddress[] = {0x84, 0xF3, 0xEB, 0xCC, 0x03, 0xEF};
// uint8_t newMACAddress[] = {0x50, 0xD2, 0xF5, 0x2F, 0x21, 0x69}; // MAC router
// uint8_t newMACAddress[] = {0x04, 0xe5, 0x98, 0x32, 0xD0, 0x1E}; // MAC Android
// uint8_t newMACAddress[] = {0x34, 0x13, 0xE8, 0xC6, 0x4C, 0x42}; // MAC Komputer

```

Gambar 4.11 Data MAC address pada lingkungan fisik

```
WiFiClient client;

ESP8266WebServer server(80);
```

Gambar 4.12 Deklarasi object dari kelas WiFiClient dan ESP8266WebServer pada TCP port 80 untuk memprogram NodeMCU sebagai client dan menjalankan server web untuk kebutuhan penelitian HTTP GET

```
void do_ping(uint8_t a, uint8_t b, uint8_t c, uint8_t d) {
  IPAddress remoteIP(a, b, c, d); // Target

  Serial.print("Pinging ");
  Serial.println(remoteIP);

  int pingTime = Ping.ping(remoteIP);
  pingTime = Ping.averageTime();

  if (pingTime > 0) {
    Serial.print("Ping successful! Latency: ");
    Serial.print(pingTime);
    Serial.println(" ms");
  } else {
    Serial.println("Ping failed!");
  }
}
```

Gambar 4.13 deklarasi fungsi untuk melakukan ping ke alamat IP yang dituju

```
void do_curl(const char * url) {
  HTTPClient http;

  Serial.print("HTTP GET request to: ");
  Serial.println(url);

  http.begin(client, url); // Specify the URL

  int httpCode = http.GET(); // Perform the GET request
S
  if (httpCode > 0) { // Check for a successful request
    Serial.printf("HTTP GET request successful. Response code:
    %d\n", httpCode);

    String payload = http.getString(); // Get the response
    payload (HTML content)

    Serial.println("Response (first 10 characters):");
    Serial.println(payload.substring(0, 10));
  } else {
    Serial.printf("HTTP GET request failed. Error code: %d\n",
    httpCode);
  }

  http.end(); // Close the connection
}
```

Gambar 4.14 deklarasi fungsi curl untuk melakukan HTTP GET ke server web dari perangkat yang dituju

```
// run HTTP server
void http_server() {
  server.send(200, "text/plain", "Hello World!");
}

```

Gambar 4.15 deklarasi fungsi untuk menjalankan server web pada perangkat NodeMCU, yang menampilkan pesan "Hello, world!"

```
void setup() {
  Serial.begin(115200);
  delay(10);
  Serial.print(" Connect to : ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("….");
  }
  Serial.print("\n");
  Serial.print("IP address : ");
  Serial.print(WiFi.localIP());
  Serial.print("\n");
  Serial.print("Connect to : ");
  Serial.println(ssid);
  Serial.print("\n");
  Serial.println(WiFi.macAddress());
  wifi_set_macaddr(STATION_IF, & newMACAddress[0]);
  Serial.print("\n");
  Serial.println(WiFi.macAddress());

  server.on("/", http_server);
  server.begin();

  Serial.println("HTTP server started.");

}
```

Gambar 4.16 deklarasi setup pada Arduino IDE

```
void loop() {

  // Uji HTTP server
  server.handleClient();

  /*
    // Uji ping dan HTTP GET
    do_ping(192,168,1,11);
    do_ping(192,168,1,1);
    do_ping(192,168,1,74);
    do_ping(192,168,1,99);
    do_curl("http://192.168.1.11"); // Alamat NodeMCU
    do_curl("http://192.168.1.1"); // Alamat router
    do_curl("http://192.168.1.74:1111"); // Alamat Android
    do_curl("http://192.168.1.99"); // Alamat komputer
    // traceroute(192,168,1,1);
    // traceroute(192,168,1,99);
    delay(1000);

    */
}

```

Gambar 4.17 deklarasi loop pada Arduino IDE, karena do_ping() dan do_curl bersifat synchronous, maka server.handleClient() tidak dapat berjalan bersamaan. Untuk melakukan pengujian ping dan curl, maka blok komentar tersebut dihapus dan dipindahkan ke server.handleClient()

BAB V
HASIL DAN PEMBAHASAN

5.1 Percobaan sementara pada lingkungan fisik

Dengan konfigurasi berikut, perangkat penguji akan mengambil MAC Address dari perangkat lain yang terhubung. Sebelum tersambung, MAC Address perangkat penguji adalah default atau diacak terlebih dahulu, kemudian setelah tersambung dilakukan scanning untuk mendapatkan MAC Address dari perangkat lain dalam jaringan yang sama.

Hasil sementara yang diperoleh adalah adanya fenomena peningkatan ping, dimana pada keadaan normal, dimana tidak ada MAC Address yang sama, ping berkisar dari dua hingga lima milisekon. Pada keadaan dimana MAC Address NodeMCU disamakan dengan MAC Address dari router, terjadi lonjakan latency, dimana ping berkisar dari 230 hingga 296 milisekon, bahkan respon ping perangkat komputer menuju router menjadi *request timed out* (RTO).

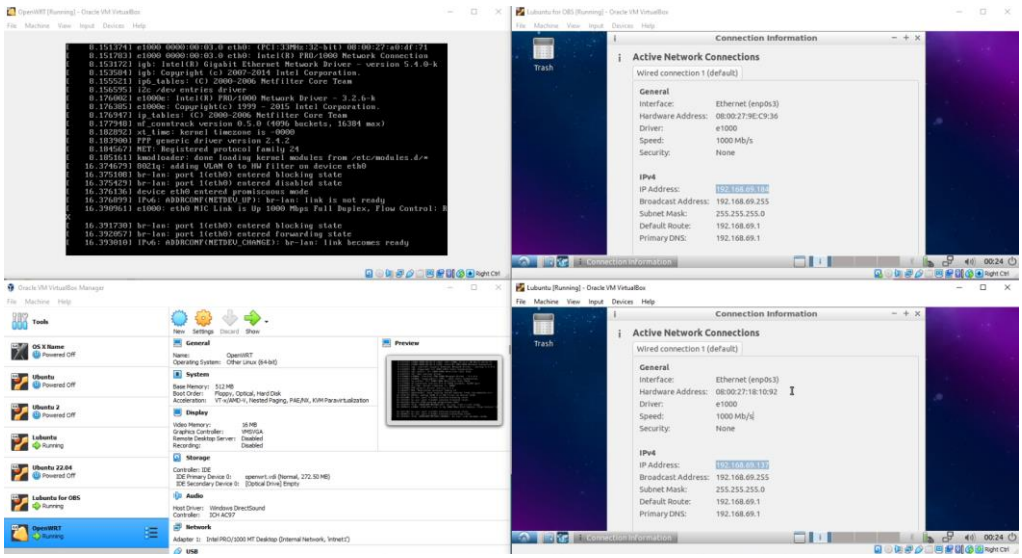
MAC NodeMCU	MAC Computer	MAC Handphone	MAC Router	avg Ping Computer, Router (ms)	avg Ping Computer, Handphone (ms)	avg Ping Handphone, Router (ms)	avg Ping Handphone, Computer (ms)
0	34:13:E8:C6:4C:42	80:35:C1:3E:3B:30	30:42:40:1E:09:8C	5	4	2	2
30:42:40:1E:09:8C	34:13:E8:C6:4C:43	80:35:C1:3E:3B:31	30:42:40:1E:09:8C	RTO	296	230	
80:35:C1:3E:3B:31	34:13:E8:C6:4C:44	80:35:C1:3E:3B:32	30:42:40:1E:09:8C	3	RTO		RTO
0	34:13:E8:C6:4C:45	80:35:C1:3E:3B:33	F4:F5:DB:D2:D2:FB	89	103	40	103
F4:F5:DB:D2:D2:FB	34:13:E8:C6:4C:46	80:35:C1:3E:3B:34	F4:F5:DB:D2:D2:FB	80	120		

Tabel 5.1 Hasil sementara yang didapat pra-proposal

5.2 Percobaan pada lingkungan virtual dengan VirtualBox

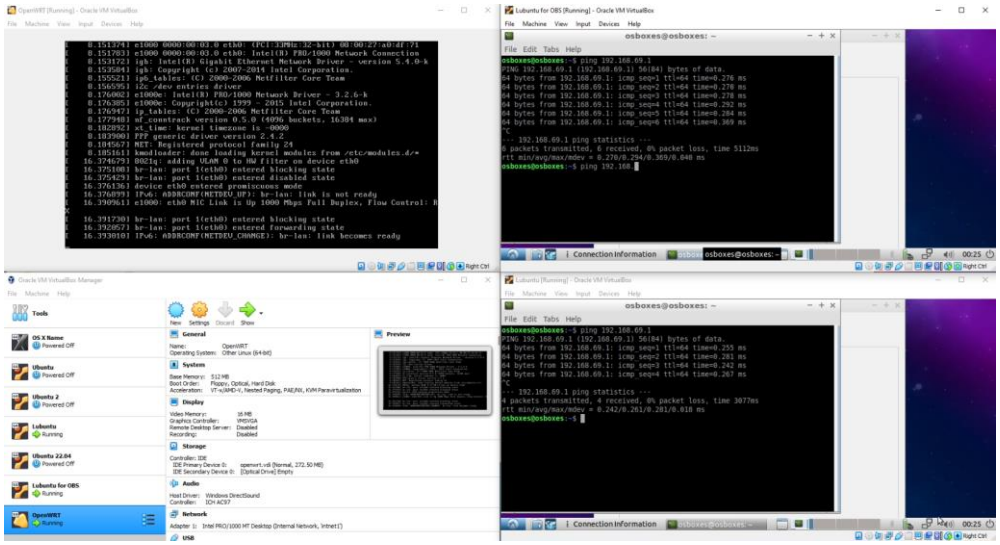
Keadaan Normal (tidak ada duplikasi alamat MAC)

Percobaan dengan VirtualBox dalam keadaan ideal, dimana semua VM, termasuk VM OpenWRT, memiliki alamat MAC yang unik. VM OpenWRT yang bertindak sebagai *router*, sekaligus server DHCP, memberikan alamat IP yang unik untuk kedua VM dengan alamat IP masing – masing 192.168.69.184 dan 192.168.69.137 dengan alamat IP *gateway* 192.168.69.1, yaitu VM OpenWRT itu sendiri.



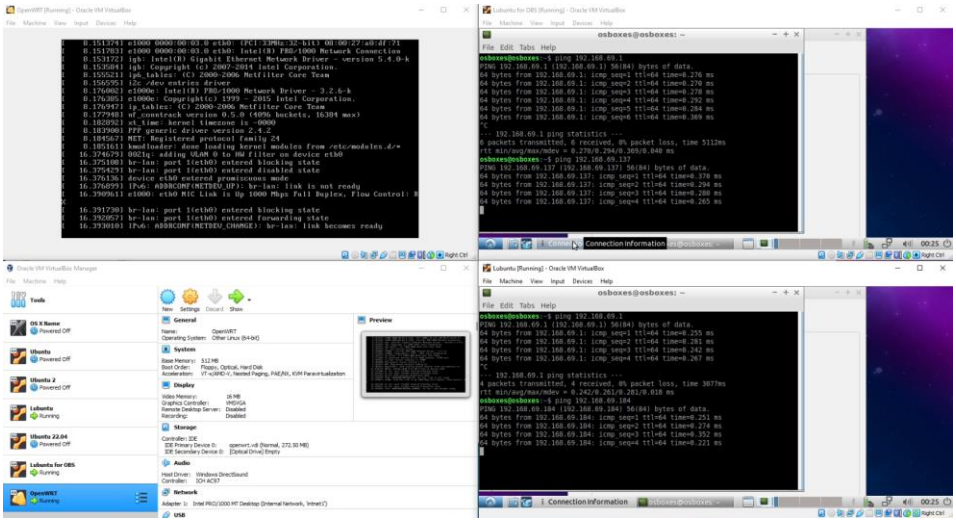
Gambar 5.1 Pengujian pada VirtualBox pada keadaan normal

Parameter yang diuji adalah *ping*. VM pertama dan kedua melakukan pengujian *ping* ke VM OpenWRT yang bertindak sebagai *gateway*. Hasilnya, *ping* kedua VM membutuhkan waktu kurang dari 1 milisekon.



Gambar 5.2 Ping ke gateway

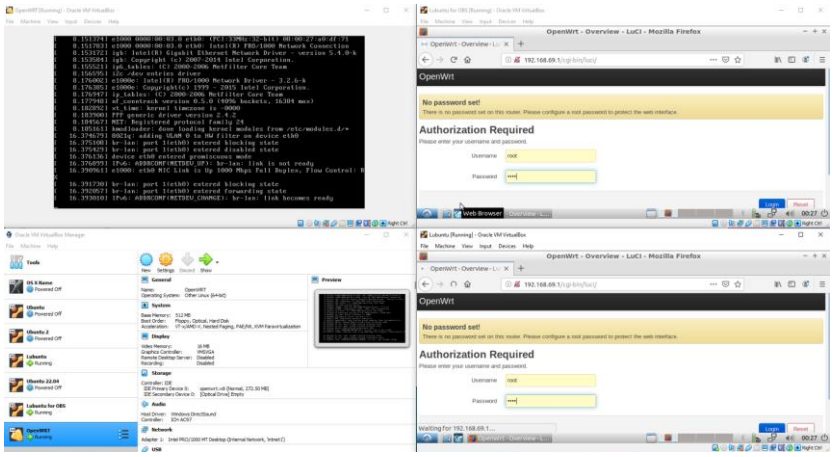
Kemudian, VM pertama dengan IP 192.168.69.184 melakukan *ping* ke VM kedua, dengan IP 192.168.69.137 dan sebaliknya. Hasilnya adalah kedua VM mampu memberikan *latency* dibawah 1 milisekon, baik dari VM 192.168.69.184 menuju VM 192.168.69.137, maupun dari VM 192.168.69.137 ke VM 192.168.69.184.



Gambar 5.3 Ping ke masing – masing VM satu sama lain

Kemudian, dilakukan pengujian HTTP GET ke VM OpenWRT dengan menggunakan *browser* Mozilla Firefox yang ada pada sistem operasi Lubuntu.

Dengan memasukkan alamat *host* atau IP *gateway*, maka hal tersebut setara dengan HTTP GET dengan menggunakan cURL. Hasilnya adalah login page *gateway* dapat diperoleh dengan baik.



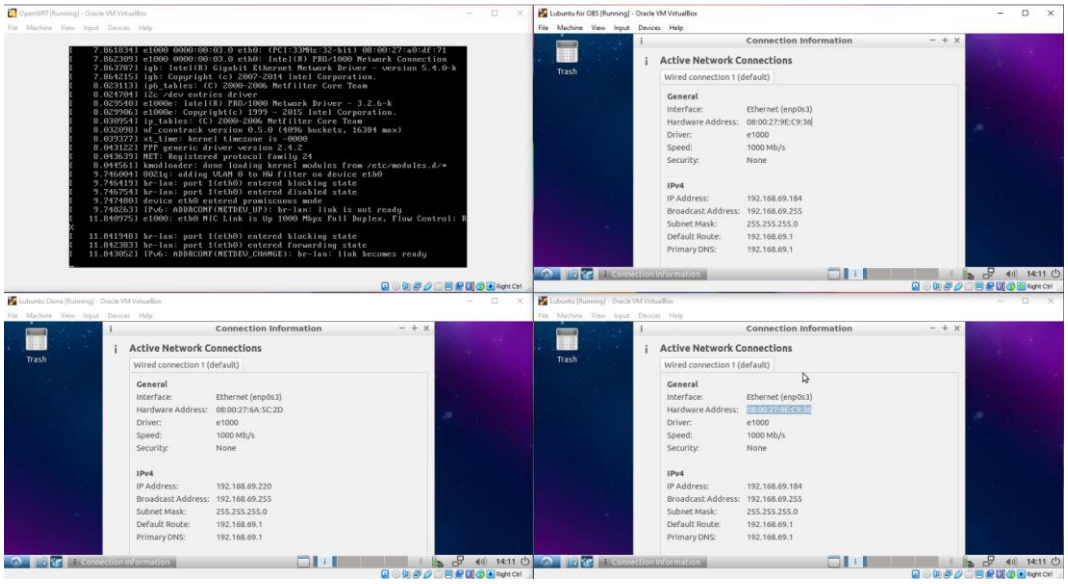
Gambar 5.4 HTTP GET request menuju login page OpenWRT

MAC VM Variabel bebas	MAC VM 1	MAC VM Pengamat	MAC router	avg Ping MAC VM variabel bebas ke Router (ms)	avg Ping MAC VM variabel bebas ke VM 1 (ms)	avg Ping MAC VM variabel bebas ke VM pengamat (ms)
08:00:27:D8:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.2	0.2	0.2
MAC VM Variabel bebas	MAC VM 1	MAC VM Pengamat	MAC router	avg HTTP GET VM variabel begbas ke Router (ms)		
08:00:27:D8:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	Berhasil		

Tabel 5.2 Tabel hasil penelitian keadaan normal

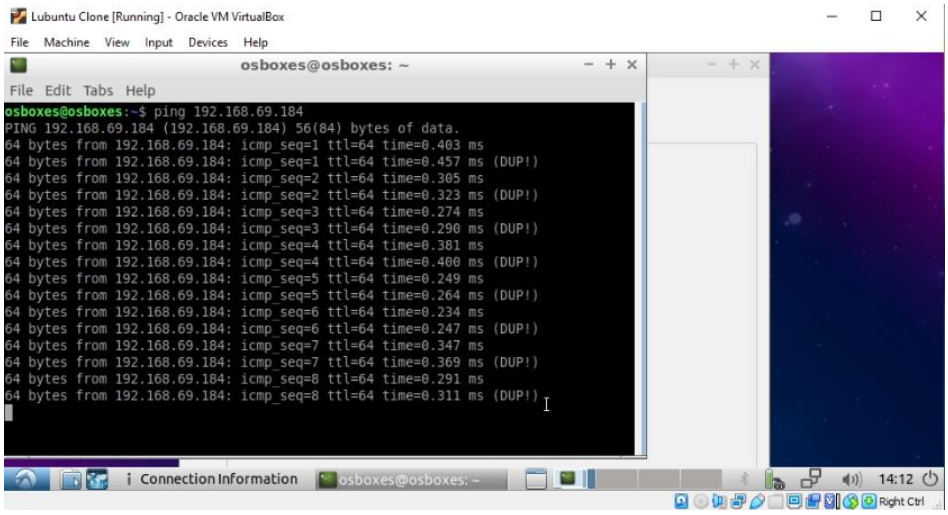
Keadaan dua VM dengan alamat MAC sama dan satu VM pengamat

Percobaan berikutnya dengan VirtualBox dalam keadaan dua buah VM dengan alamat MAC yang sama, hanya MAC VM yang memiliki alamat MAC yang unik. Tampak server DHCP memberikan alamat IP yang sama untuk kedua VM tersebut, sehingga kedua VM memiliki alamat IP 192.168.69.184. Hanya satu VM yang memiliki satu alamat IP yang unik, yaitu 192.168.69.220.



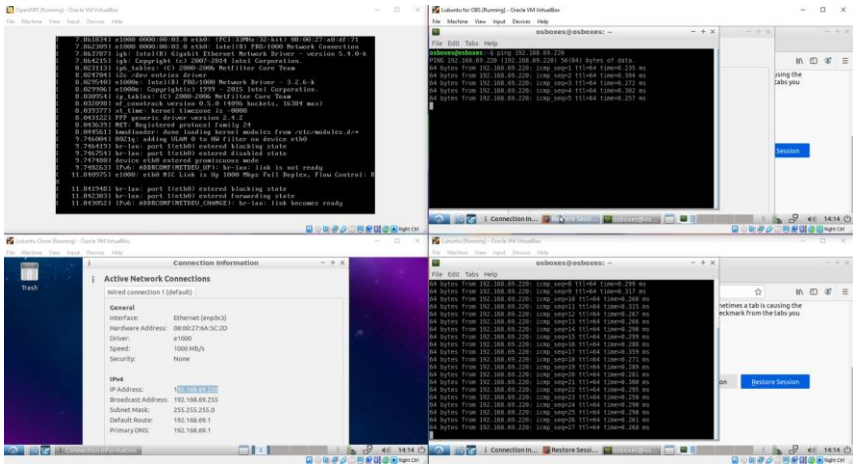
Gambar 5.5 Kedua buah VM memiliki alamat MAC yang sama

Pada saat VM pengamat melakukan *ping* ke alamat IP VM dengan alamat MAC yang sama, hasilnya adalah terjadi respon ganda dengan pesan “(DUP!)”, yang menandakan adanya *duplicate packets*, namun *latency* tetap lebih kecil dari 1 milidetik.



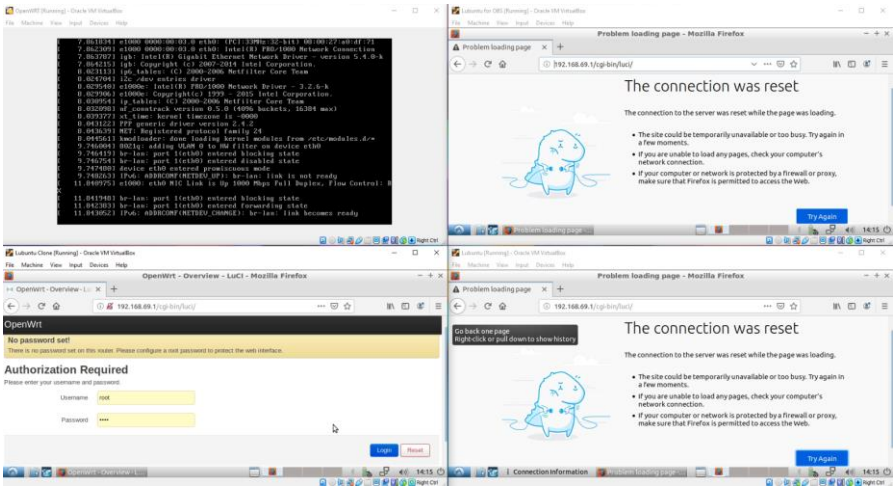
Gambar 5.6 Ping dengan respon "(DUP!)"

Kemudian, dilakukan *ping* dari masing – masing VM dengan alamat MAC *duplicate* ke VM pengamat. Hasilnya, *ping* tampak normal dan *latency* keduanya bernilai kurang dari 1 milidetik.



Gambar 5.7 Ping dari masing – masing VM duplicate ke VM pengamat

Selanjutnya, dilakukan percobaan HTTP GET ke *webpage* OpenWRT dengan alamat 192.168.69.1 dari masing – masing VM yang terdiri dari dua VM dengan alamat MAC yang sama dan satu VM pengamat. Hasilnya adalah hanya VM pengamat yang dapat menampilkan *webpage* OpenWRT, sedangkan kedua VM *duplicate* tidak dapat menampilkannya, serta terdapat pesan "*The Connection was reset*".



Gambar 5.8 HTTP GET pada VM pengamat berhasil namun gagal pada VM duplicate

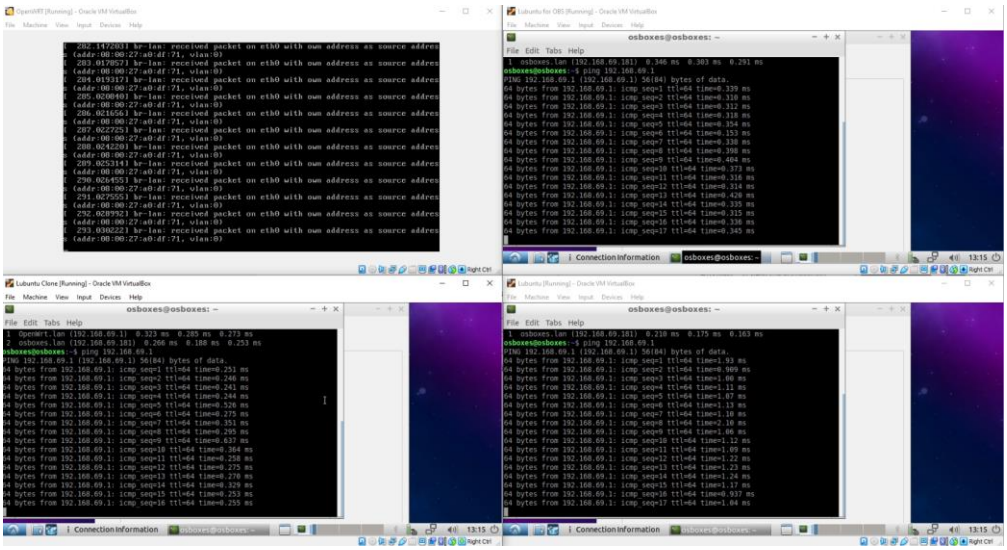
MAC VM Variabel bebas	MAC VM 1	MAC VM Pengamat	MAC router	avg Ping MAC VM variabel bebas ke Router (ms)	avg Ping MAC VM variabel bebas ke VM 1 (ms)	avg Ping MAC VM variabel bebas ke VM pengamat (ms)
08:00:27: D8:98:53	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.2	0.2	0.2
08:00:27: 9E:C9:36	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.2	0.2 (DUP!)	0.2
MAC VM Variabel bebas	MAC VM 1	MAC VM Pengamat	MAC router	avg HTTP GET VM variabel begbas ke Router (ms)		
08:00:27: D8:98:53	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	Berhasil		
08:00:27: 9E:C9:36	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	Gagal		

Tabel 5.3 Perbandingan tabel hasil penelitian keadaan VM variabel bebas duplicate dengan VM pengamat

Keadaan VM duplicate dengan VM OpenWRT

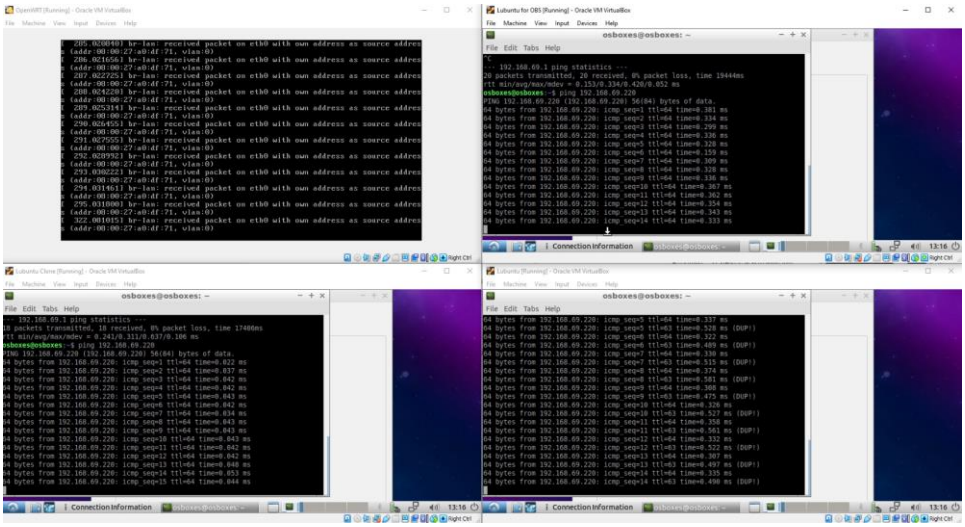
Percobaan berikutnya dengan VirtualBox dalam keadaan salah satu alamat MAC VM memiliki alamat yang sama dengan alamat MAC *router* OpenWRT. Alamat MAC VM yang akan disamakan dengan alamat MAC OpenWRT adalah 08:00:27:D8:98:53, diganti menjadi alamat MAC OpenWRT dengan nilai 08:00:27:A0:DF:71.

Pengujian *ping* dilakukan menuju *router* OpenWRT dengan alamat IP 192.168.69.1. Semua VM berhasil melakukan *ping* ke VM OpenWRT, namun pada salah satu VM dengan alamat MAC yang sama dengan VM OpenWRT, nilai *latency* berada 0.909 ms hingga 2.10 ms, berbeda dengan *latency* kedua VM lain menuju VM OpenWRT dengan alamat MAC yang unik.



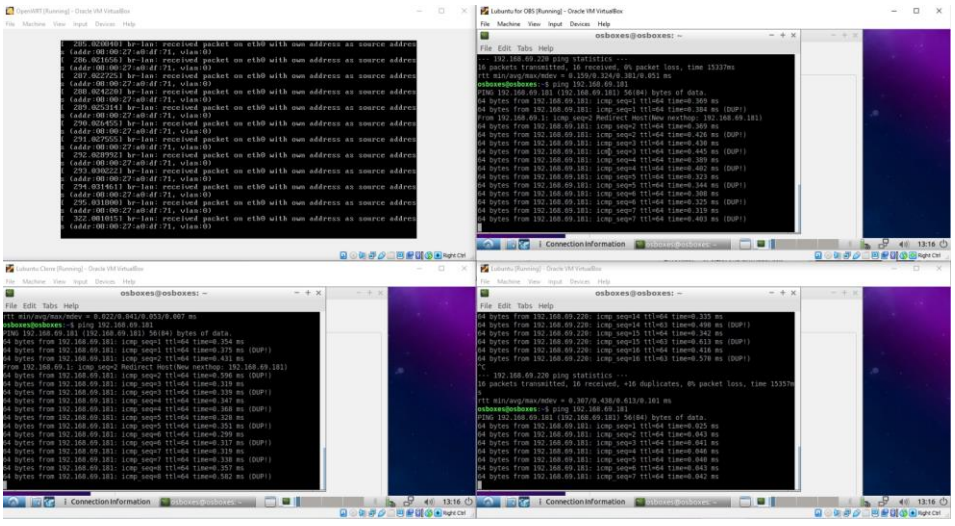
Gambar 5.9 Pengujian ping menuju router OpenWRT

Pengujian *ping* selanjutnya adalah menuju VM dengan alamat MAC unik dengan alamat IP 192.168.69.220. Semua perangkat berhasil melakukan *ping* dengan *latency* dibawah 1 ms, namun pada VM *duplicate* terdapat respon ganda dengan pesan “DUP!”.



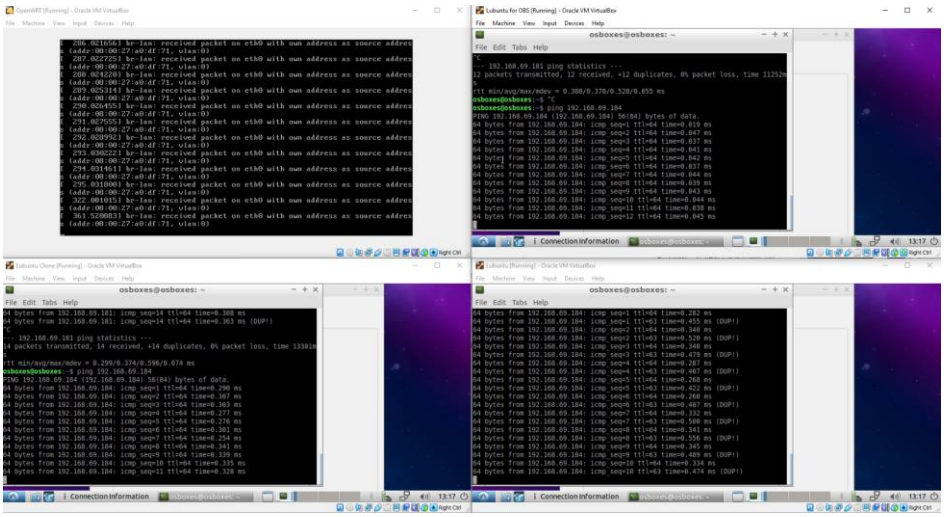
Gambar 5.10 Pengujian ping menuju VM unik pertama

Pengujian *ping* selanjutnya adalah menuju VM dengan alamat MAC *duplicate* yang sama dengan alamat MAC OpenWRT dengan alamat IP 192.168.69.181. Kedua VM dengan alamat MAC yang unik berhasil melakukan *ping* dengan *latency* dibawah 1ms namun terdapat respon ganda "DUP!" dan terdapat pesan "*From 192.168.69.1: icmp_seq=2 Redirect Host(New nexthop: 192.168.69.181)*" pada paket ICMP dengan nomor *sequence* 2.



Gambar 5.11 Pengujian ping menuju VM duplicate

Pengujian *ping* yang terakhir adalah menuju VM dengan alamat MAC unik kedua dengan alamat IP 192.168.69.184. Sama seperti pada percobaan *ping* pada VM pertama, semua perangkat berhasil melakukan *ping* dengan *latency* dibawah 1 ms, namun pada VM *duplicate* terdapat respon ganda dengan pesan “DUP!”.



Gambar 5.10 Pengujian ping menuju VM unik kedua

MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	avg Ping variabel bebas ke Router (ms)	avg Ping MAC VM 1 ke Router (ms)	avg Ping MAC VM 2 bebas ke Router (ms)
08:00:27:D0:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.2	0.2	0.2
08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	1.2	0.2	0.2

MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	avg Ping variabel bebas ke VM Variabel bebas (ms)	avg Ping MAC VM 1 ke VM Variabel bebas (ms)	avg Ping MAC VM 2 ke VM Variabel bebas (ms)
08:00:27:D0:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.2	0.2	0.2
08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0	0.3 (DUP!)	0.3 (DUP!)

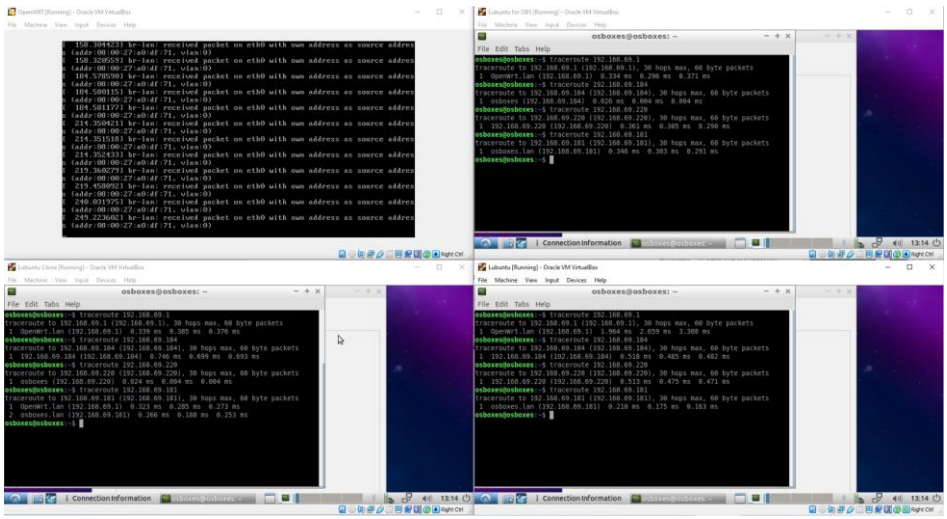
MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	avg Ping MAC variabel bebas ke VM 1 (ms)	avg Ping MAC VM 1 ke VM 1 (ms)	avg Ping MAC MAC VM 2 bebas ke VM 1 (ms)
08:00:27:D0:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.2	0.2	0.2
08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.2 (DUP!)	0	0.2

MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	avg Ping MAC variabel bebas ke VM 2 (ms)	avg Ping MAC VM 1 ke VM 2 (ms)	avg Ping MAC MAC VM 2 bebas ke VM 2 (ms)
08:00:27:D0:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.2	0.2	0.2
08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	0.3 (DUP!)	0.2	0

Tabel 5.4 Pengujian ping pada masing – masing VM

Pengujian selanjutnya adalah *traceroute*. *Traceroute* dilakukan menuju alamat IP masing – masing VM yang berbeda. Semua *traceroute* berhasil dilakukan,

namun pada pengujian *traceroute* yang pertama, dimana masing – masing VM melakukan *traceroute* menuju VM OpenWRT dengan alamat IP 192.168.69.1, VM yang memiliki alamat MAC yang sama dengan OpenWRT memiliki *latency* yang relatif lebih besar dibandingkan kedua VM lainnya, dimana VM *duplicate* memiliki *latency* sebesar 1.964 – 3.308 ms, berbeda dengan kedua VM lainnya yang memiliki *latency* sebesar 0.296 – 0.385 ms. Kemudian, pada pengujian *traceroute* menuju VM OpenWRT, terdapat dua buah *hop* pada salah satu VM unik, yaitu VM OpenWRT pada *hop* pertama, dan VM *duplicate* pada *hop* kedua. Pada keadaan normal dalam suatu jaringan yang sama, *hop* hanya terdapat satu buah, dengan alamat IP *hop* tujuan pada respon *traceroute*. Lalu, pada saat *traceroute* dilakukan menuju masing – masing perangkat itu sendiri pada masing – masing perangkat, kedua perangkat unik memiliki *latency* yang sangat kecil, dengan nilai 0.004 – 0.024 ms, namun pada VM *duplicate* nilai *latency* nya bernilai 0.163 – 0.210 ms.



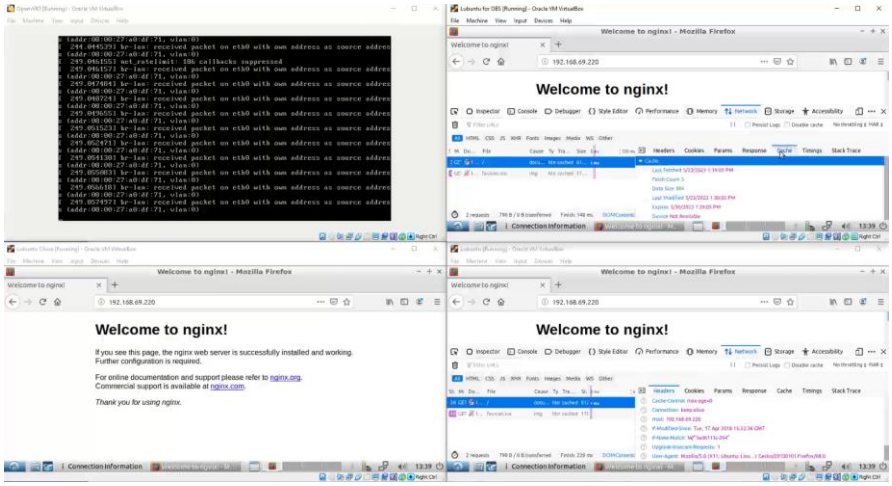
Gambar 5.11 Pengujian traceroute pada masing – masing VM

MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	traceroute MAC variabel bebas ke Router (ms)	traceroute MAC VM 1 ke Router (ms)	traceroute MAC VM 2 bebas ke Router (ms)		MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	traceroute MAC variabel bebas ke VM Variabel bebas (ms)	traceroute MAC VM 1 ke VM Variabel bebas (ms)	traceroute MAC VM 2 ke VM Variabel bebas (ms)
08:00:27: D8:98:53	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.3	0.3	0.3		08:00:27: D8:98:53	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0	0.3	0.3
08:00:27: A0:DF:71	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	3.3	0.3	0.3		08:00:27: A0:DF:71	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.2	0.5 (2 hop)	0.3

MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	traceroute MAC variabel bebas ke VM 1 (ms)	avg Ping MAC VM 1 ke VM 1 (ms)	traceroute MAC VM 2 bebas ke VM 1 (ms)		MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	traceroute MAC variabel bebas ke VM 2 (ms)	traceroute MAC VM 1 ke VM 2 (ms)	traceroute MAC MAC VM 2 bebas ke VM 2 (ms)
08:00:27: D8:98:53	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.3	0	0.3		08:00:27: D8:98:53	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.3	0.3	0
08:00:27: A0:DF:71	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.5	0	0.2		08:00:27: A0:DF:71	08:00:27: 6A:5C:2D	08:00:27: 9E:C9:36	08:00:27: A0:DF:71	0.5	0.4	0.4

Tabel 5.5 Hasil pengujian traceroute pada masing – masing VM

Pengujian HTTP GET dilakukan dengan adanya server web Nginx yang berjalan pada masing – masing VM pada port TCP 80. Semua permintaan HTTP GET berhasil ke masing – masing VM dengan *latency* 1ms, baik menuju masing – masing server web Nginx dan *webpage* OpenWRT.

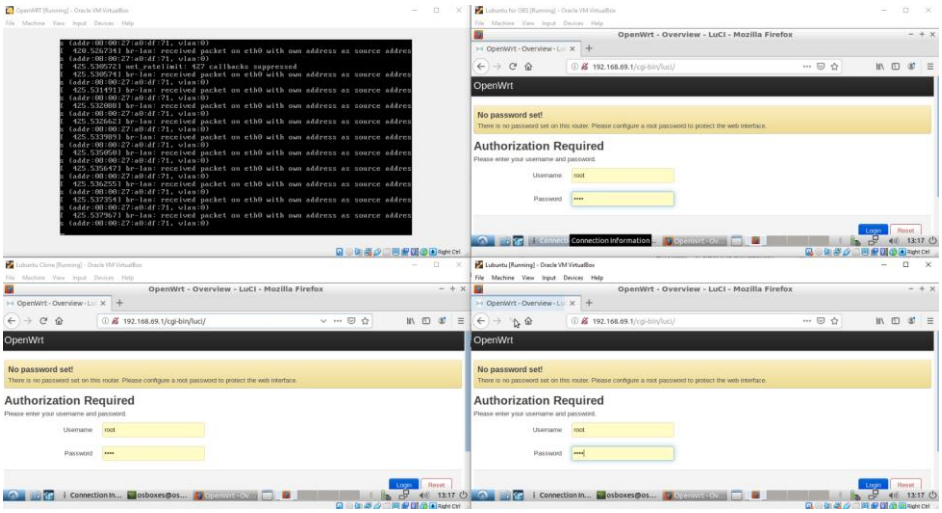


Gambar 5.12 Pengujian HTTP GET dengan server web Nginx

MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	HTTP GET MAC variabel bebas ke Router (ms)	traceroute MAC VM 1 ke Router (ms)	traceroute MAC VM 2 bebas ke Router (ms)		MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	HTTP GET MAC variabel bebas ke VM Variabel bebas (ms)	HTTP GET MAC VM 1 ke VM Variabel bebas (ms)	HTTP GET MAC VM 2 ke VM Variabel bebas (ms)
08:00:27:D8:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil		08:00:27:D8:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil
08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil		08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil

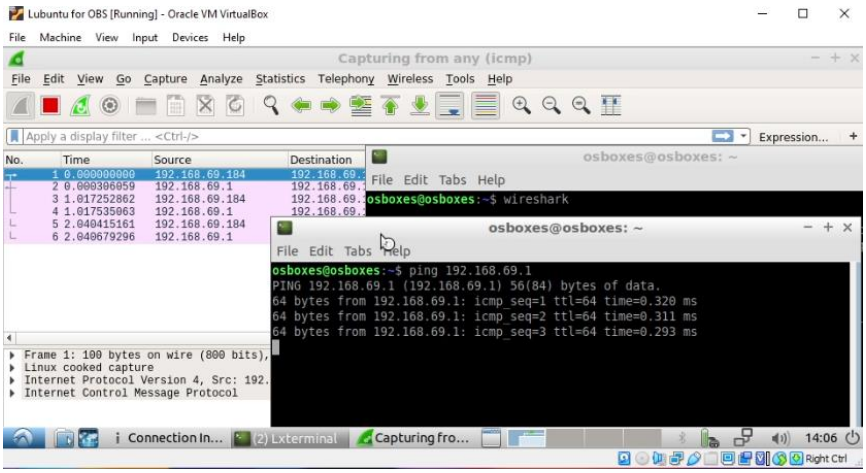
MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	HTTP GET MAC variabel bebas ke VM 1 (ms)	HTTP GET MAC VM 1 ke VM 1 (ms)	HTTP GET MAC VM 2 bebas ke VM 1 (ms)		MAC VM Variabel bebas	MAC VM 1	MAC VM 2	MAC router	HTTP GET MAC variabel bebas ke VM 2 (ms)	HTTP GET MAC VM 1 ke VM 2 (ms)	HTTP GET MAC MAC VM 2 bebas ke VM 2 (ms)
08:00:27:D8:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil		08:00:27:D8:98:53	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil
08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil		08:00:27:A0:DF:71	08:00:27:6A:5C:2D	08:00:27:9E:C9:36	08:00:27:A0:DF:71	berhasil	berhasil	berhasil

Tabel 5.6 Hasil pengujian HTTP GET pada masing – masing VM

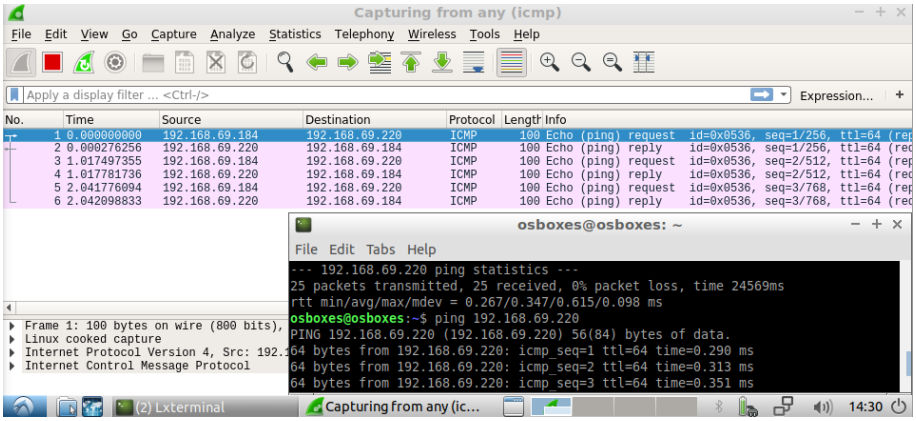


Gambar 5.13 Pengujian HTTP GET menuju webpage OpenWRT

Analisa melalui program Wireshark dilakukan pada VM pengamat dengan alamat IP 192.168.69.184. VM pengamat tidak dapat mengamati *traffic* ICMP pada VM lain, karena desain *internal network* pada VirtualBox yang berupa *switch*, dan bukan *hub*. Maka, analisa melalui Wireshark hanya dapat dilakukan pada salah satu VM yang akan melakukan pengujian *ping* dan *traceroute*. Percobaan pertama dilakukan dengan melakukan *ping* ke VM OpenWRT dan VM lainnya dengan alamat IP 192.168.69.220. Hasilnya, tampak bahwa dalam satu kali *ping* terdapat satu kali respon *pong*, sehingga untuk 3 kali *ping* dengan nomor *icmp_seq=3*, total payload yang dikirim dan diterma adalah 6.

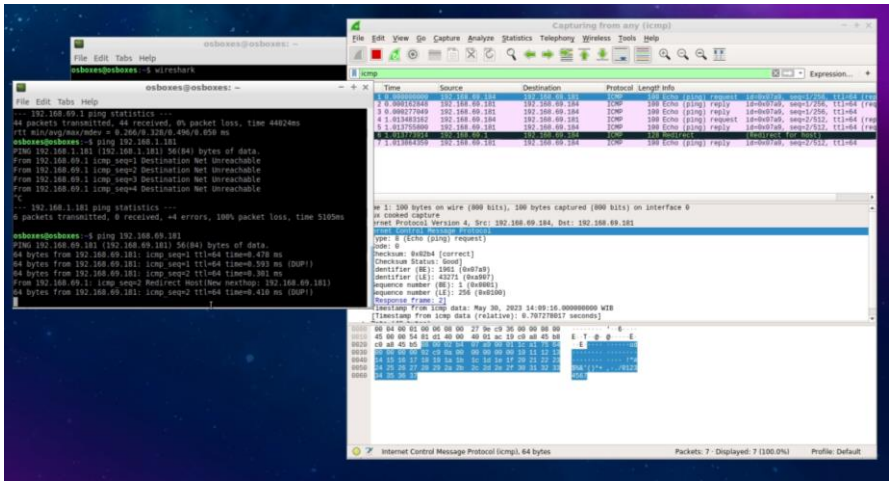


Gambar 5.14 Pengujian ping dari VM pengamat ke VM OpenWRT



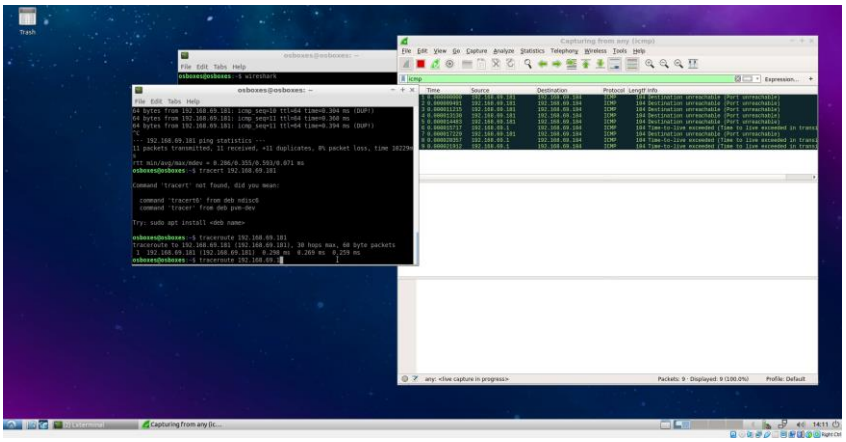
Gambar 5.15 Pengujian ping dari VM pengamat ke VM lainnya

Selanjutnya, dilakukan percobaan *ping* dari VM pengamat ke VM *duplicate*. Hasilnya, terdapat total 7 buah *payload* hanya pada percobaan *ping* kedua, dengan nomor *icmp_seq=2.* Terdapat dua buah *payload pong* setelah *ping* dengan nomor *icmp_seq=1*, dan satu buah payload *pong* satu buah *payload redirect*, dan satu buah *payload pong* setelah *redirect* pada *ping* dengan nomor *icmp_seq=2.*

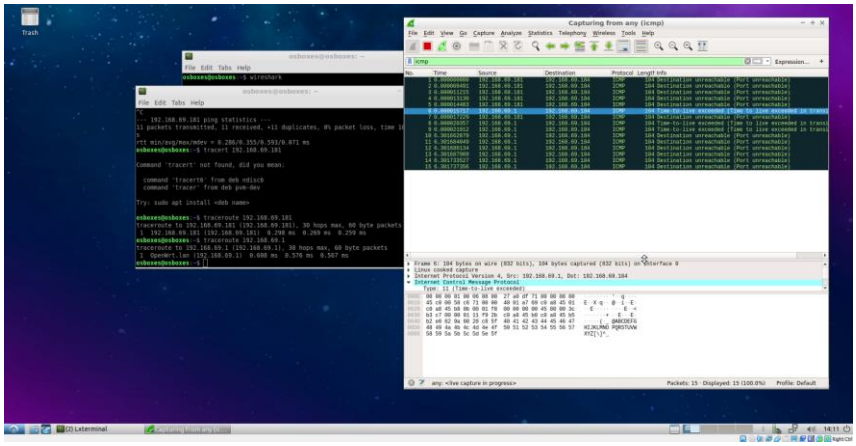


Gambar 5.16 Pengujian ping dari VM pengamat ke VM duplicate

Selanjutnya, dilakukan pengamatan *traceroute* pada Wireshark menuju VM *duplicate* dengan alamat IP 192.168.69.181. *Payload traceroute* tampak lebih banyak dibandingkan *traceroute* ke VM lain dan juga pada keadaan normal, dan juga terdapat *payload "Time-to-live exceeded"* dari VM OpenWRT.



Gambar 5.17 Pengujian ping dari VM pengamat ke VM duplicate



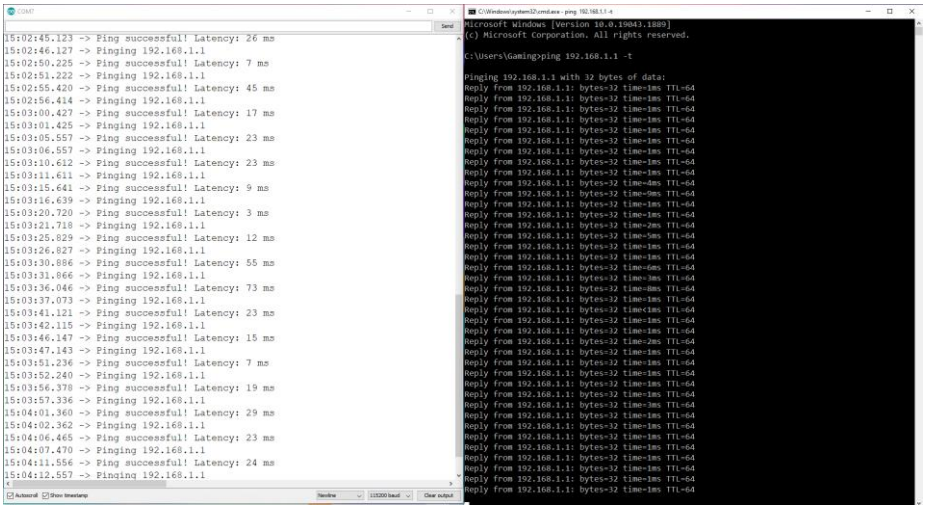
Gambar 5.18 Pengujian ping dari VM pengamat ke VM OpenWRT

5.3 Percobaan pada lingkungan fisik

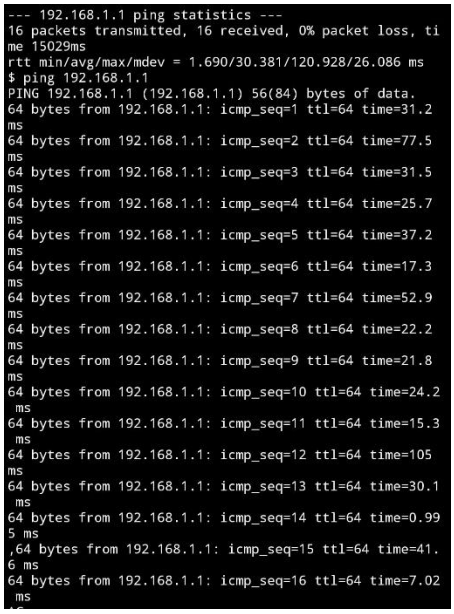
Keadaan Normal (tidak ada duplikasi alamat MAC)

Percobaan dilakukan dengan *router* Mi Router 4A, dengan *client* NodeMCU sebagai variabel bebas (perangkat yang dapat mengubah alamat MAC), client kedua yaitu perangkat Android, dan komputer Windows dengan NIC Intel Wi-Fi 6 AX200 yang berjalan pada mode 802.11ac (pada frekuensi 5GHz) melalui percobaan meliputi *ping*, HTTP GET *request*, dan *traceroute*, serta analisis Wireshark.

Pada keadaan normal, ping NodeMCU ke *router* adalah berkisar antara 3ms – 58ms, *ping* komputer ke *router* adalah 1ms, serta *ping* dari perangkat Android ke *router* adalah 7 – 105ms.



Gambar 5.19 Ping NodeMCU (kiri) dan ping komputer (kanan) menuju router

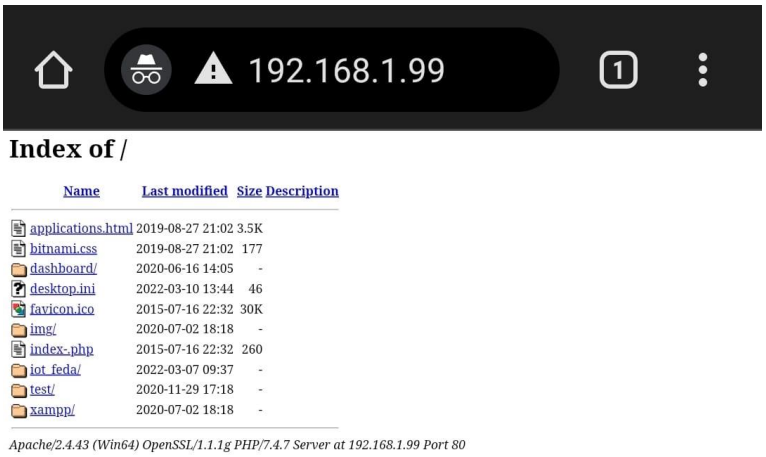


Gambar 5.20 Ping Android menuju router

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Router (ms)	avg Ping komputer ke Router (ms)	avg Ping Android ke Router (ms)		MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Komputer (ms)	avg Ping komputer ke Komputer (ms)	avg Ping Android ke Komputer (ms)
84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69	8	1	30		84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69	17	0	28
MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke NodeMCU (ms)	avg Ping komputer ke NodeMCU (ms)	avg Ping Android ke NodeMCU (ms)		MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Android (ms)	avg Ping komputer ke Android (ms)	avg Ping Android ke Android (ms)
84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69					84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69	266	288	0

Tabel 5.7 Hasil pengujian ping pada masing – masing perangkat

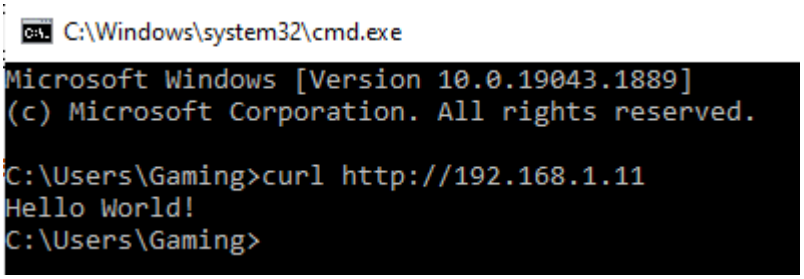
Pengujian GET dilakukan dengan menjalankan server Apache pada komputer dan pada *homepage router*. Baik server Apache dan *homepage router* dapat menampilkan halaman server, baik melalui cURL maupun aplikasi *browser*.



Gambar 5.21 HTTP GET menuju server Apache pada komputer

```
15:46:14.864 -> HTTP GET request to: http://192.168.1.1
15:46:14.960 -> HTTP GET request successful. Response code: 200
15:46:15.198 -> Response (first 10 characters):
15:46:15.198 ->
15:46:15.198 -> HTTP GET request to: http://192.168.1.99
15:46:15.292 -> HTTP GET request successful. Response code: 200
15:46:15.340 -> Response (first 10 characters):
15:46:15.340 -> <!DOCTYPE
```

Gambar 5.22 HTTP GET dari NodeMCU menuju router dan komputer



Gambar 5.23 HTTP GET dari komputer ke NodeMCU

Gambar 5.24 HTTP GET dari komputer ke Android

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Router (ms)	HTTP GET komputer ke Router (ms)	HTTP GET Android ke Router (ms)		MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Komputer (ms)	HTTP GET komputer ke Komputer (ms)	HTTP GET Android ke Komputer (ms)
84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69	Berhasil	Berhasil	Berhasil		84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69	Berhasil	Berhasil	Berhasil

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke NodeMCU (ms)	HTTP GET komputer ke NodeMCU (ms)	HTTP GET Android ke NodeMCU (ms)		MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Android (ms)	HTTP GET komputer ke Android (ms)	HTTP GET Android ke Android (ms)
84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69		Berhasil	Berhasil		84:F3:EB: CC:03:EF	34:13:E8: C6:4C:42	04:E5:98: 32:D0:1E	50:D2:F5: 2F:21:69	Berhasil	Berhasil	Berhasil

Tabel 5.8 Hasil pengujian HTTP GET pada masing – masing perangkat

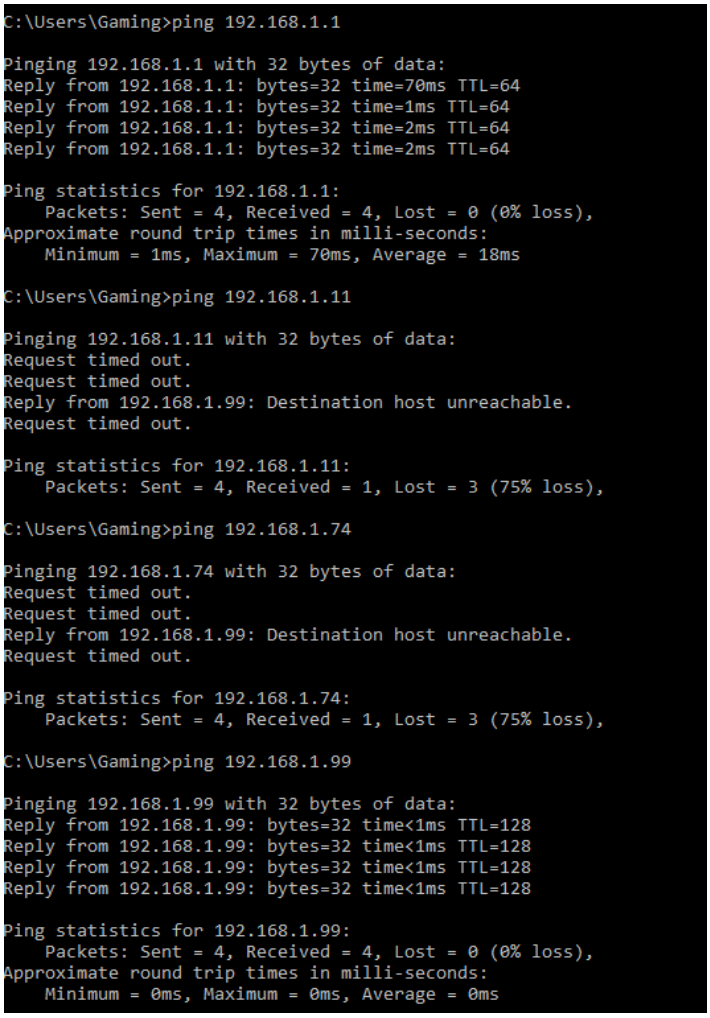
Keadaan MAC NodeMCU *duplicate* dengan *router*

Pengujian *ping* pada kondisi MAC NodeMCU *duplicate* dengan router menunjukkan hasil, dimana NodeMCU tidak dapat melakukan *ping* dan HTTP GET ke semua perangkat dalam suatu jaringan.

```
20:23:53.573 -> 84:F3:EB:CC:03:EF
20:23:53.573 ->
20:23:53.573 -> 50:D2:F5:2F:21:69
20:23:53.573 -> HTTP server started.
20:23:53.573 -> Pinging 192.168.1.11
20:23:58.564 -> Ping failed!
20:23:58.564 -> Pinging 192.168.1.1
20:24:03.610 -> Ping failed!
20:24:03.610 -> Pinging 192.168.1.74
20:24:08.613 -> Ping failed!
20:24:08.613 -> Pinging 192.168.1.99
20:24:13.662 -> Ping failed!
20:24:13.662 -> HTTP GET request to: http://192.168.1.11
20:24:13.662 -> HTTP GET request failed. Error code: -1
20:24:13.662 -> HTTP GET request to: http://192.168.1.1
20:24:13.662 -> HTTP GET request failed. Error code: -1
20:24:13.662 -> HTTP GET request to: http://192.168.1.74
20:24:13.662 -> HTTP GET request failed. Error code: -1
20:24:13.662 -> HTTP GET request to: http://192.168.1.99
20:24:13.662 -> HTTP GET request failed. Error code: -1
```

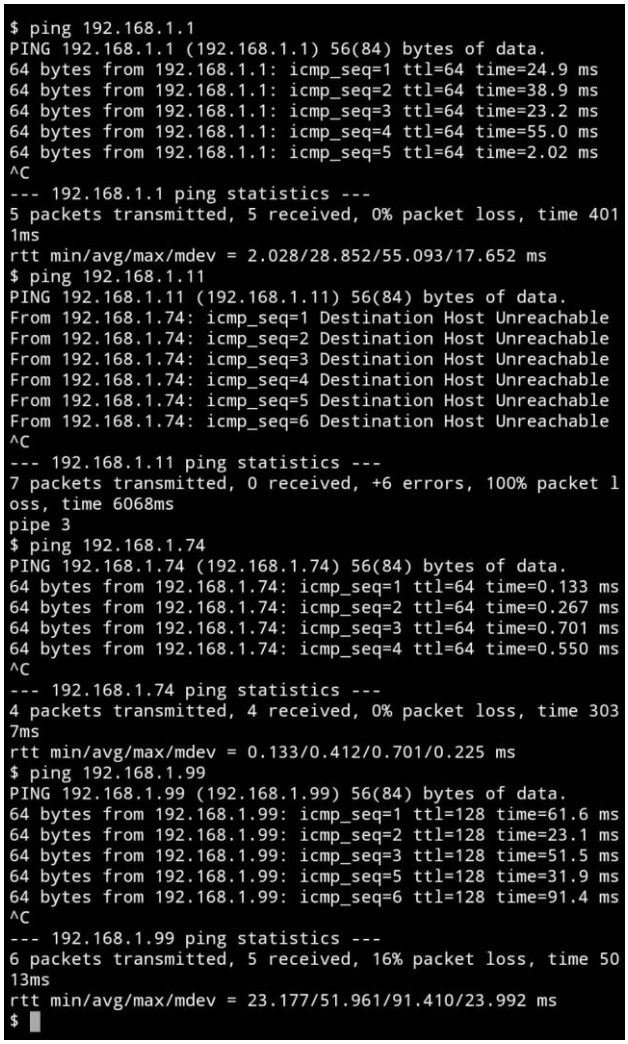
Gambar 5.25 Ping dan HTTP GET dari NodeMCU ke semua perangkat

Sedangkan *ping* dari komputer hanya berhasil menuju *router* dengan *latency* rata – rata yang lebih besar.



Gambar 5.25 Ping dari Komputer ke semua perangkat

Kemudian *ping* dari Android hanya berhasil menuju *router* dan komputer.



Gambar 5.26 Ping dari Android ke semua perangkat

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Router (ms)	avg Ping komputer ke Router (ms)	avg Ping Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	8	1	30
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	18	28

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Komputer (ms)	avg Ping komputer ke Komputer (ms)	avg Ping Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	17	0	28
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	RTO	51

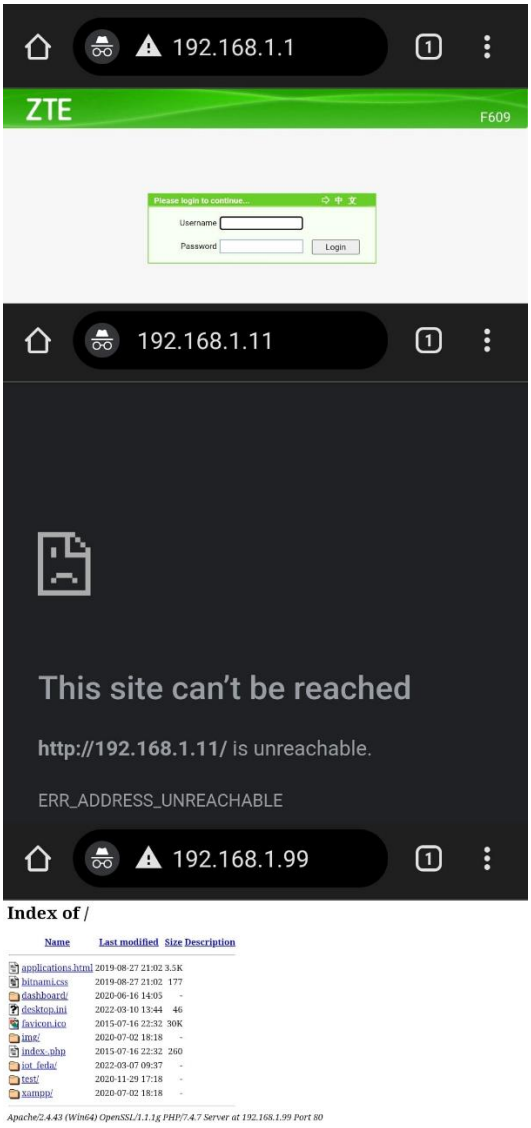
MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke NodeMCU (ms)	avg Ping komputer ke NodeMCU (ms)	avg Ping Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69			
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69			

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Android (ms)	avg Ping komputer ke Android (ms)	avg Ping Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	266	288	0
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	RTO	0

Tabel 5.9 Hasil ping pada masing – masing perangkat

Untuk HTTP GET, tidak seperti pada lingkungan VirtualBox, terjadi kegagalan untuk kasus HTTP GET dari NodeMCU ke semua perangkat, komputer ke NodeMCU dan Android, dan Android ke NodeMCU.

Gambar 5.27 HTTP GET dari komputer ke semua perangkat



Gambar 5.28 HTTP GET dari Android ke semua perangkat

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Router (ms)	HTTP GET komputer ke Router (ms)	HTTP GET Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Berhasil

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Komputer (ms)	HTTP GET komputer ke Komputer (ms)	HTTP GET Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Berhasil

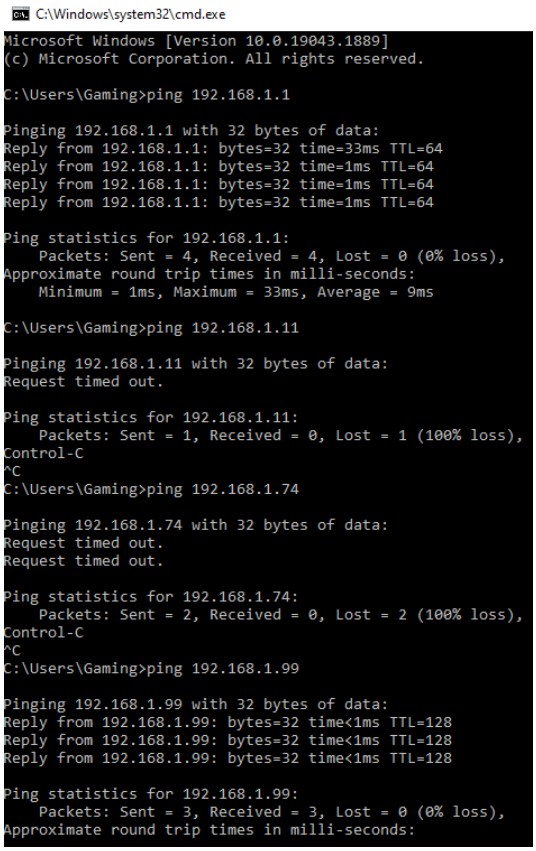
MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke NodeMCU (ms)	HTTP GET komputer ke NodeMCU (ms)	HTTP GET Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69		Gagal	Gagal

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Android (ms)	HTTP GET komputer ke Android (ms)	HTTP GET Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
50:D2:F5:2F:21:69	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Gagal	Berhasil

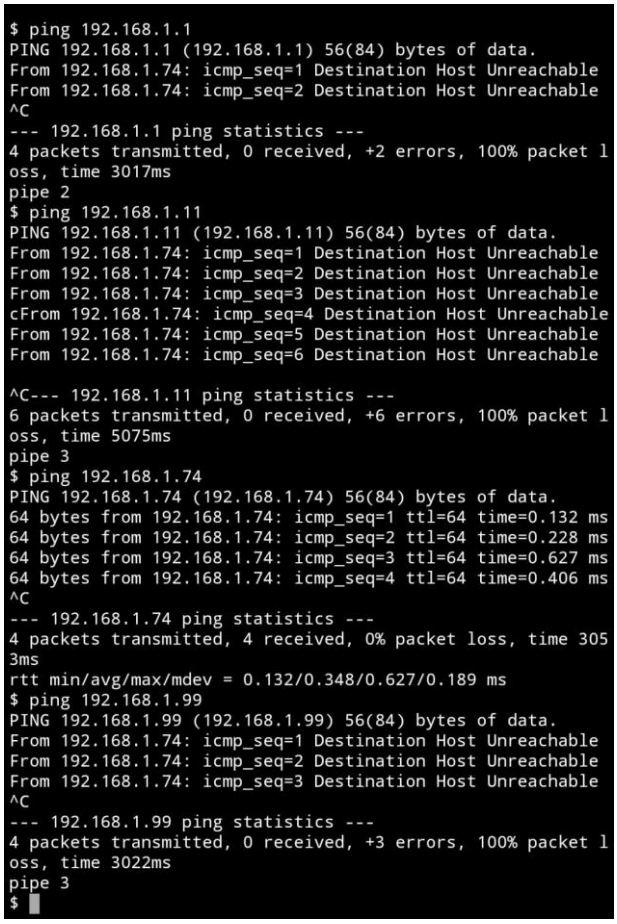
Tabel 5.10 Hasil HTTP GET pada masing – masing perangkat

Keadaan MAC NodeMCU *duplicate* dengan *client* (Android)

Untuk *ping* dalam keadaan MAC NodeMCU sama dengan perangkat Android, *ping* menunjukkan RTO pada *ping* dari NodeMCU ke Android dan *ping* yang relatif lebih besar dari data keadaan normal, dari komputer menuju Android, dan dari Android menuju *router* dan komputer.



Gambar 5.29 Ping dari komputer ke semua perangkat



Gambar 5.30 Ping dari Android ke semua perangkat

```
21:53:35.371 -> Pinging 192.168.1.11
21:53:40.410 -> Ping failed!
21:53:40.410 -> Pinging 192.168.1.1
21:53:44.457 -> Ping successful! Latency: 59 ms
21:53:44.457 -> Pinging 192.168.1.74
21:53:49.500 -> Ping failed!
21:53:49.500 -> Pinging 192.168.1.99
21:53:53.689 -> Ping successful! Latency: 37 ms
```

Gambar 5.31 Ping dari NodeMCU ke semua perangkat

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Router (ms)	avg Ping komputer ke Router (ms)	avg Ping Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	8	1	30
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	59	9	RTO

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Komputer (ms)	avg Ping komputer ke Komputer (ms)	avg Ping Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	17	0	28
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	37	0	RTO

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke NodeMCU (ms)	avg Ping komputer ke NodeMCU (ms)	avg Ping Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69			
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69			

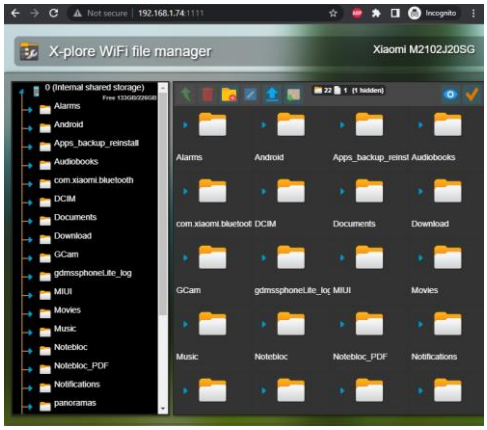
MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Android (ms)	avg Ping komputer ke Android (ms)	avg Ping Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	266	288	0
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	RTO	0

Tabel 5.11 Hasil ping pada masing – masing perangkat

Lalu untuk HTTP GET, terjadi kegagalan pada saat melakukan HTTP GET dari NodeMCU ke semua perangkat, komputer menuju NodeMCU, dan Android menuju NodeMCU dan komputer.

```
22:02:02.582 -> HTTP GET request to: http://192.168.1.11
22:02:07.768 -> HTTP GET request failed. Error code: -1
22:02:07.768 -> HTTP GET request to: http://192.168.1.1
22:02:13.013 -> HTTP GET request failed. Error code: -1
22:02:13.013 -> HTTP GET request to: http://192.168.1.74
22:02:18.183 -> HTTP GET request failed. Error code: -1
22:02:18.230 -> HTTP GET request to: http://192.168.1.99
22:02:23.419 -> HTTP GET request failed. Error code: -1
```

Gambar 5.32 HTTP GET dari NodeMCU ke semua perangkat



Gambar 5.32 HTTP GET dari komputer ke Android

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Router (ms)	HTTP GET komputer ke Router (ms)	HTTP GET Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Berhasil

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Komputer (ms)	HTTP GET komputer ke Komputer (ms)	HTTP GET Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Gagal

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke NodeMCU (ms)	HTTP GET komputer ke NodeMCU (ms)	HTTP GET Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69		Gagal	Gagal

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Android (ms)	HTTP GET komputer ke Android (ms)	HTTP GET Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
04:E5:98:32:D0:1E	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Berhasil

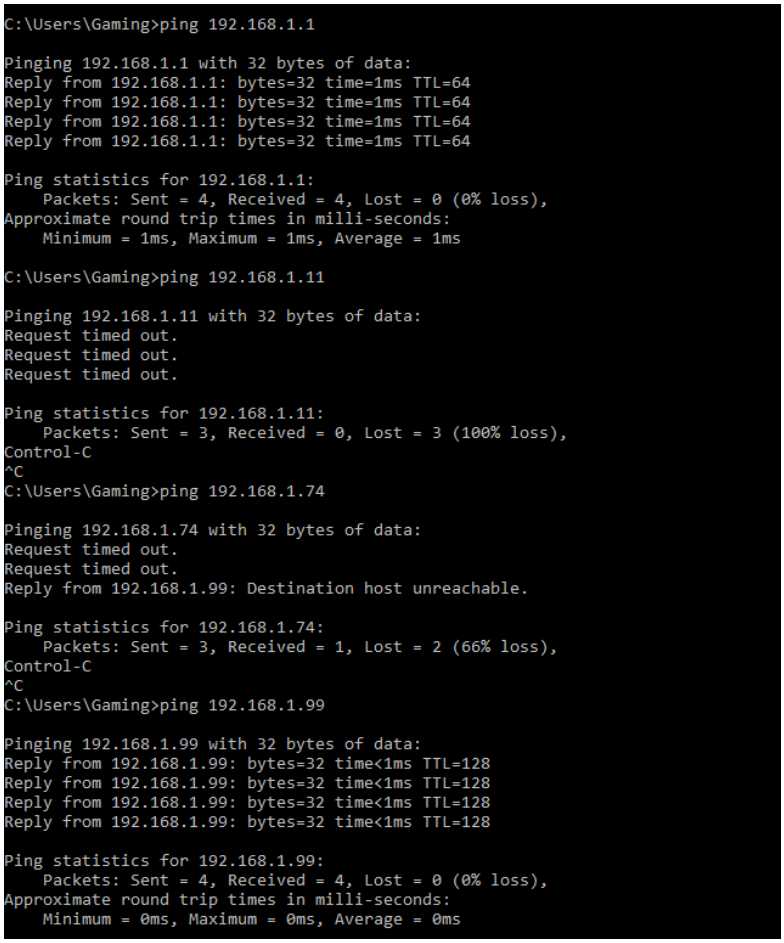
Tabel 5.11 Hasil HTTP GET pada masing – masing perangkat

Keadaan MAC NodeMCU *duplicate* dengan *client* (Komputer)

Untuk *ping* dalam keadaan MAC NodeMCU sama dengan perangkat komputer, *ping* menunjukkan RTO pada *ping* dari NodeMCU ke semua perangkat, dari komputer menuju Android. *Ping* perangkat Android menuju *router* dan komputer relatif lebih besar dibandingkan pada keadaan normal.

```
23:56:09.667 -> HTTP server started.
23:56:09.667 -> Pinging 192.168.1.11
23:56:14.672 -> Ping failed!
23:56:14.672 -> Pinging 192.168.1.1
23:56:19.737 -> Ping failed!
23:56:19.737 -> Pinging 192.168.1.74
23:56:24.794 -> Ping failed!
23:56:24.794 -> Pinging 192.168.1.99
23:56:29.897 -> Ping failed!
23:56:29.897 -> HTTP GET request to: http://192.168.1.11
23:56:35.129 -> HTTP GET request failed. Error code: -1
23:56:35.129 -> HTTP GET request to: http://192.168.1.1
23:56:35.177 -> HTTP GET request successful. Response code: 200
23:56:35.318 -> Response (first 10 characters):
23:56:35.318 ->
23:56:35.318 -> HTTP GET request to: http://192.168.1.74:1111
23:56:40.525 -> HTTP GET request failed. Error code: -1
23:56:40.525 -> HTTP GET request to: http://192.168.1.99
23:56:45.671 -> HTTP GET request failed. Error code: -1
```

Gambar 5.33 Ping dan HTTP GET dari NodeMCU ke semua perangkat



Gambar 5.34 Ping dari komputer ke semua perangkat

```
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=28.3 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=14.7 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=369 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=1.62 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=4.36 ms
^C
--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 400
6ms
rtt min/avg/max/mdev = 1.629/83.747/369.657/143.262 ms
$ ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
From 192.168.1.74: icmp_seq=1 Destination Host Unreachable
From 192.168.1.74: icmp_seq=2 Destination Host Unreachable
From 192.168.1.74: icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.1.11 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet l
oss, time 5062ms
pipe 3
$ ping 192.168.1.74
PING 192.168.1.74 (192.168.1.74) 56(84) bytes of data.
64 bytes from 192.168.1.74: icmp_seq=1 ttl=64 time=0.105 ms
64 bytes from 192.168.1.74: icmp_seq=2 ttl=64 time=0.382 ms
64 bytes from 192.168.1.74: icmp_seq=3 ttl=64 time=0.423 ms
64 bytes from 192.168.1.74: icmp_seq=4 ttl=64 time=0.466 ms
^C
--- 192.168.1.74 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 304
4ms
rtt min/avg/max/mdev = 0.105/0.344/0.466/0.141 ms
$ ping 192.168.1.99
PING 192.168.1.99 (192.168.1.99) 56(84) bytes of data.
64 bytes from 192.168.1.99: icmp_seq=1 ttl=255 time=142 ms
64 bytes from 192.168.1.99: icmp_seq=2 ttl=255 time=53.3 ms
64 bytes from 192.168.1.99: icmp_seq=3 ttl=255 time=124 ms
64 bytes from 192.168.1.99: icmp_seq=4 ttl=255 time=291 ms
^C
--- 192.168.1.99 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 300
5ms
rtt min/avg/max/mdev = 53.329/153.034/291.512/86.636 ms
$
```

Gambar 5.35 Ping dari Android ke semua perangkat

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Router (ms)	avg Ping komputer ke Router (ms)	avg Ping Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	8	1	30
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	1	83

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Komputer (ms)	avg Ping komputer ke Komputer (ms)	avg Ping Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	17	0	28
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	0	153

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke NodeMCU (ms)	avg Ping komputer ke NodeMCU (ms)	avg Ping Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69			
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69			

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	avg Ping NodeMCU ke Android (ms)	avg Ping komputer ke Android (ms)	avg Ping Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	266	288	0
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	RTO	RTO	0

Tabel 5.12 Hasil ping pada masing – masing perangkat

Untuk *HTTP GET* dalam keadaan MAC NodeMCU sama dengan perangkat komputer, NodeMCU gagal melakukan HTTP GET ke komputer dan Android, komputer gagal melakukan HTTP GET ke NodeMCU dan Android, dan Android gagal melakukan HTTP GET ke NodeMCU, namun pada kejadian, dimana Android melakukan HTTP GET ke komputer, respons yang ditampilkan adalah “Hello, world!”, yang merupakan *server* dari NodeMCU.

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Router (ms)	HTTP GET komputer ke Router (ms)	HTTP GET Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Komputer (ms)	HTTP GET komputer ke Komputer (ms)	HTTP GET Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Gagal

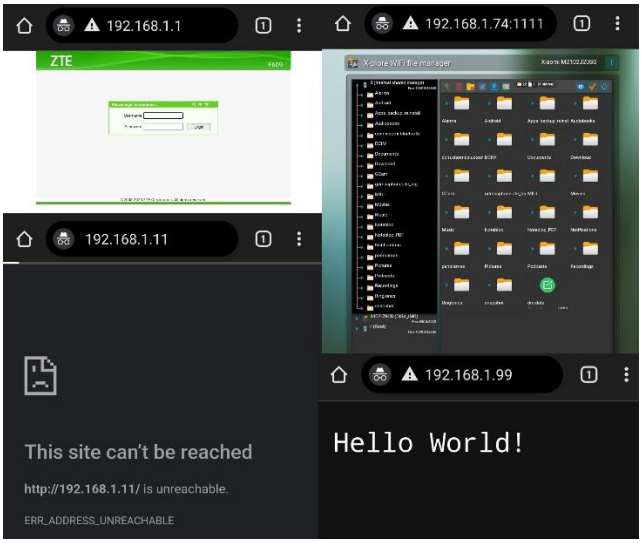
MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke NodeMCU (ms)	HTTP GET komputer ke NodeMCU (ms)	HTTP GET Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69		Gagal	Gagal

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Android (ms)	HTTP GET komputer ke Android (ms)	HTTP GET Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Gagal	Berhasil

```
submitForm.appendChild(randomElement);
submitForm.appendChild(checkTokenElement);
if(getObj("Frm_Validatecode") != null)
{
var validCodeElement = createHiddenInput("Validatecode", getObj("Frm_Validatecode").value);
submitForm.appendChild(validCodeElement);
}
submitForm.submit();
}
</script>

C:\Users\Gaming>curl http://192.168.1.11
^C
C:\Users\Gaming>curl http://192.168.1.74
^C
C:\Users\Gaming>curl http://192.168.1.74:1111
^C
C:\Users\Gaming>curl http://192.168.1.99
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
 <head>
  <title>Index of /</title>
 </head>
 <body>
<h1>Index of /</h1>
```

Gambar 5.36 HTTP GET komputer ke semua perangkat



Gambar 5.37 HTTP GET Android ke semua perangkat, dimana Android ke komputer mendapatkan respon dari NodeMCU

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Router (ms)	HTTP GET komputer ke Router (ms)	HTTP GET Android ke Router (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Komputer (ms)	HTTP GET komputer ke Komputer (ms)	HTTP GET Android ke Komputer (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Berhasil	Gagal

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke NodeMCU (ms)	HTTP GET komputer ke NodeMCU (ms)	HTTP GET Android ke NodeMCU (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69		Gagal	Gagal

MAC Variabel bebas (NodeMCU)	MAC Komputer	MAC Android	MAC router	HTTP GET NodeMCU ke Android (ms)	HTTP GET komputer ke Android (ms)	HTTP GET Android ke Android (ms)
84:F3:EB:CC:03:EF	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Berhasil	Berhasil	Berhasil
34:13:E8:C6:4C:42	34:13:E8:C6:4C:42	04:E5:98:32:D0:1E	50:D2:F5:2F:21:69	Gagal	Gagal	Berhasil

Tabel 5.13 Hasil HTTP GET pada masing – masing perangkat, dimana Android ke komputer mendapatkan respon dari NodeMCU

BAB VI
KESIMPULAN

6.1 Kesimpulan

Berdasarkan pengujian dan pengamatan yang dilakukan, serta analisis hasil yang telah diperoleh, maka dapat disimpulkan bahwa:

1. Penelitian MAC *address collision* menghasilkan karakteristik anomali yang berbeda – beda pada lingkungan virtual VirtualBox dan percobaan pada lingkungan fisik.
2. Pada pengujian *ping*, fenomena MAC *address collision* dapat ditemukan dengan adanya pesan "DUP!" pada *response* dari *ping* pada sistem operasi Linux yang berjalan pada lingkungan VirtualBox, bertambahnya *latency* dari keadaan normal untuk penelitian baik dalam VirtualBox maupun lingkungan fisik, bahkan gagalnya *ping*, yang menggunakan parameter *request timed out* (RTO) terjadi pada lingkungan fisik.
3. Pada pengujian *traceroute* di VirtualBox, terjadi kenaikan nilai *latency*, serta beberapa kali melalui 2 buah *hop*, dimana pada keadaan normal seharusnya tidak terjadi.
4. Pada pengujian HTTP GET, semua perangkat pada VirtualBox berhasil menampilkan webpage, baik melalui *browser* maupun cURL, namun pada lingkungan fisik hanya beberapa saja yang berhasil, dan adanya respon HTTP GET yang tidak seharusnya ditampilkan perangkat yang dituju, terutama pada saat melakukan HTTP GET dari Android ke komputer, dimana respon yang di dapat adalah respon dari NodeMCU.

DAFTAR PUSTAKA

Alasdair Allan “List of MAC addresses with vendors identities”. [Daring] Tersedia dari: https://gist.github.com/aallan/b4bb86db86079509e6159810ae9bd3e4 [Diakses:07/04/2022]

Android 6.0 changes. Tersedia dari : https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html, 2015. [Diakses:07/04/2022]

Cardenas, Edgar D. “MAC Spoofing--An Introduction”. GIAC Security Essentials Certification. SANS Institute 8 Februari 2013.

Cisco. “Configuring Port Security”. 14 November 2015.

F. Baker (June 1995). Baker, F (ed.). Requirements for IP Version 4 Routers. p. 52. RFC 1812.

Institute of Electrical and Electronics Engineers (September 2004). “Overview and Guide to the IEEE 802 LMSC” (PDF). Tersedia dari : https://grouper.ieee.org/groups/802/802%20overview.pdf [Diakses:15/07/2022]

Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, Frank Piessens, iMinds-Distrinet, KU Leuven, Univ Lyon, INSA Lyon, Inria, CITI, France. “Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms”.

Oracle Corporation. “Changelog for VirtualBox 6.1”. 19 April 2022. Tersedia dari : https://www.virtualbox.org/wiki/Changelog-6.1 [Diakses:15/07/2022]

Xerox Corporation. “Xerox System Integration Standard 098404 - Authentication Protocol” (PDF). 1984.

Junade Ali, Vladimir Dyo “Practical Hash-Based Anonymity for MAC Addresses”. Cloudflare Inc, London, UK. University of Bedfordshire, Luton, UK. 2020

Jean-François Determe, Sophia Azzagnuni, François Horlin, Philippe De Doncker “MAC Address Anonymization for Crowd Counting”. BEAMS-EE, Université Libre de Bruxelles, 1050 Brussels, Belgium. OPERA Wireless Communications Group, Université Libre de Bruxelles, 1050 Brussels, Belgium. 2020

LAMPIRAN

Data Penelitian	https://go.gabrielkheisa.xyz/data_penelitian
Repository penelitian	https://repo.gabrielkheisa.xyz/gabrielkheisa/skripsi
Program NodeMCU	https://repo.gabrielkheisa.xyz/gabrielkheisa/skripsi/src/branch/master/source_nodemcu.cpp
IJEIS	https://go.gabrielkheisa.xyz/ijeis
Dokumentasi VirtualBox	https://go.gabrielkheisa.xyz/dokumentasi_virtualbox