SKRIPSI

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

RESEARCH ON THE EFFECT OF MAC ADDRESS COLLISION ON DEVICES THAT CAN CHANGE 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
2023

SKRIPSI

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

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

Diajukan untuk memenuhi salah satu syarat memperoleh derajat Sarjana Elektronika dan Instrumentasi

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
2023

HALAMAN PENGESAHAN

SKRIPSI

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

Telah dipersiapkan dan disusun oleh

Gabriel Possenti Kheisa Drianasta

19/442374/PA/19123

Mengetahui,
a.n. Dekan FMIPA UGM
Wakil Dekan Bidang Pendidikan, Pengajaran dan Kemahasiswaan

Prof. Drs. Roto, M.Eng., Ph.D.
NIP. 196711171993031020

Telah dipertahankan di depan Tim Penguji

pada tanggal 18 September 2023

Susunan Tim Penguji

Drs. Abdul Ro`uf, M.I.Kom	Prof. Dr.techn. Ahmad Ashari, M.I.Kom.
Ketua Penguji	Anggota Penguji
Prof. Dr. Jazi Eko Istiyanto, M.Sc.	Bakhtiar Alldino Ardi Sumbodo, S.Si., M.Cs.
Pembimbing Pertama	Pembimbing Kedua

PERNYATAAN BEBAS PLAGIASI

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, 18 September 2023

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. Bapak Bapak Drs. Abdul Ro`uf, M.I.Kom dan Prof. Dr.techn. Ahmad Ashari, M.I.Kom., selaku dosen penguji skripsi saya yang telah memberikan kritik serta saran sehingga skripsi ini menjadi lebih baik.
2. Prof. Dr. Jazi Eko Istiyanto, M.Sc. Ph.D selaku pembimbing II atas segala bimbingan, arahan, dan sarannya yang berarti dalam penyelesaian skripsi ini;
3. Bakhtiar Alldino A.S., S.Si, M.Cs selaku pembimbing I atas segala bimbingan, arahan, dan sarannya yang berarti dalam penyelesaian skripsi ini;
4. Aufaclav Zatu Kusuma Frisky, S. Si, M. Sc. selaku Dosen Pembimbing Akademik atas segala bimbingan, terutama dalam hal sistem akademis.
5. Rekan – rekan Mahasiswa Elektronika dan Instrumentasi, atas kerjasama dan bantuannya;
6. Seluruh pihak yang terlibat hingga penulis tidak bisa sebutkan satu per satu.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

INTISARI

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

Oleh
Gabriel Possenti
19/442374/PA/19123

MAC *address collision* merupakan fenomena 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 atau MAC *address randomization* 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 secara langsung pada *router* fisik dan dilakukan juga pada *virtualized environment* pada VirtualBox untuk mengetahui dampak apa yang akan terjadi dengan menganalisa variabel – variabel terikat, seperti *latency*, jumlah *hop* yang dilalui, dan keberhasilan menerima respon HTTP GET.

Hasil penelitian menunjukkan adanya peningkatan rata - rata *latency*, dari 0,2 milisekon hingga 1,2 milisekon, baik pada lingkungan virtual, maupun fisik, gagalnya konektivitas pada beberapa skenario percobaan, dan didapat respon HTTP GET yang tidak sesuai sebanyak 100% untuk 10 kali percobaan pada lingkungan fisik jika terjadi MAC *Address Collision*.

Kata kunci: *MAC address collision*, *MAC address randomization*, *duplicate*, *latency*, kegagalan koneksi

ABSTRACT

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

by
Gabriel Possenti
19/442374/PA/19123

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 with a method MAC address randomization 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 influence of MAC address collisions was carried out directly on a physical router and carried out in a virtualized environment on VirtualBox to find out what impact would occur by analyzing dependent variables, such as latency, number of hops traversed, and success in receiving an HTTP GET response.

The results of the study showed an increase in the average latency, from 0.2 milliseconds to 1.2 milliseconds, both in virtual and physical environments. The connectivity also failed in some experimental scenarios, and HTTP GET responses were not received correctly with a failure rate of invalid responses 100% for 10 times of attempts in physical environments if a MAC Address Collision occurred.

Keyword: MAC address collision, MAC address randomization, duplicate, latency, connectivity failure

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 LAN. (IEEE 802, 2004). IEEE 802 meliputi aturan – aturan atau standar untuk Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11), Bluetooth (IEEE 802.15), dan sebagainya, yang mengatur bagaimana sinyal Wi-Fi dipropagasikan, atau cara mendeteksi dan memperbaiki *error* dalam mengirimkan data. 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.

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, *et al*., 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.

Ada pula metode untuk membedakan antara perangkat dengan alamat MAC asli dengan alamat MAC palsu (*spoofing*), dengan menganalisa variabel – variabel *Channel State Information* (CSI), yang terdapat pada Wi-Fi, berupa amplitudo dan fasa sinyal, yang memiliki kemungkinan perbedaan antara perangkat asli dan *spoofing* menggunakan metode *Deep Learning* (Peng Jiang, *et al*., 2018). Metode berikut dilakukan di *physical layer* pada lapisan OSI *layer*, karena variabel *amplitude* dan *phase* dari sinyal Wi-Fi berada pada lapisan berikut. Sayangnya, metode berikut hanya bekerja pada jaringan yang menggunakan Wi-Fi berbasis modulasi OFDM, dan tidak pada jaringan LAN dengan kabel *ethernet*.

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.

1.2 Rumusan Masalah

Dikarenakan adanya masalah alamat MAC yang dapat diacak dan diubah, maka alamat MAC sudah tidak bisa disebut sebagai "*physical address*" dalam pemaknaan tradisional atau alamat fisik, dalam definisi bahwa *physical address* yang bersifat *hard-coded.* Perubahan alamat MAC tidak dapat diidentifikasi oleh MAC *filtering*, yang menyebabkan penyerang dapat menyamakan alamat MAC perangkatnya dengan perangkat lain.

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 untuk mengatasi kejadian yang tidak diinginkan, yaitu gangguan pada jaringan akibat MAC Address collision. Gangguan ini dapat

terjadi karena data aliran paket dari berbagai alat dan perangkat yang digunakan dalam penelitian ini tidak dapat diterima dengan baik oleh jaringan.

1.6 Metodologi Penelitian

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, rumusan masalah penelitian, batasan masalah penelitian, tujuan penelitian, manfaat penelitian, metodologi penelitian, serta sistematika penulisan.

BAB II: TINJAUAN PUSTAKA

Bab ini berisi tentang penelitian-penelitian terdahulu mengenai pemalsuan alamat MAC, konflik yang ditimbulkan dari adanya MAC *address collision*, dokumentasi resmi dari sistem operasi yang mendukung MAC *address randomization*, hingga penggunaan Wireshark untuk mendukung penelitian.

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* yang akan mengkaji lebih banyak mengenai alamat MAC pada OSI *layer*.

BAB IV: METODE PENELITIAN

Bab ini berisi tentang penjelasan mengenai tahapan-tahapan penelitian secara keseluruhan yang alat dan bahan, tahapan penelitian, perancangan sistem, dan program yang akan digunakan untuk analisis di lingkungan fisik.

BAB V: HASIL DAN PEMBAHASAN

Bab ini menyajikan hasil penelitian dan pembahasannya secara komprehensif, mulai dari hasil analisis dan statistik penelitian, hingga variabel-variabel yang mempengaruhi hasil pengujian dan output penelitian.

BAB VI: KESIMPULAN DAN SARAN

Bab terakhir dalam laporan penelitian ini merangkum hasil penelitian yang telah dilakukan, serta memberikan masukan untuk penelitian selanjutnya.

Kesimpulan tersebut disusun berdasarkan data dan analisis yang telah dilakukan, sehingga dapat memberikan gambaran yang jelas tentang hasil penelitian. Masukan pengembangan yang diberikan diharapkan dapat menjadi bahan pertimbangan bagi peneliti lain untuk melakukan penelitian lanjutan.

BAB II
TINJAUAN PUSTAKA

Menrut penelitian Cardenas dan Edgar D (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.

Menurut penelitian yang dilakukan oleh Tang Yong, Liu Xiaoyan (2008), MAC *address collision* dapat menyebabkan konflik pada suatu jaringan. Metode deteksi MAC *address collision*, atau deteksi alamat MAC yang sama pada satu jaringan dapat dilakukan dengan mengirim *idle frame* dari perangkat pertama ke perangkat kedua, berserta data nilai fitur, bisa berupa nama perangkat, dan dikonversi ke nilai numerik. Apabila nilai fitur berbeda, maka perangkat kedua akan melaporkan adanya MAC *address collision*. Metode ini dilakukan pada topologi jaringan berbentuk cincin atau *ring network*.

Menurut dokumentasi Android Open Source Project Changes (2015.), 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 (2022) merilis dokumentasi bahwa pada sistem operasi Android 10, *MAC Address Randomization* diaktifkan secara bawaan pada mode *client, SoftAP*, dan *Wi-Fi Direct*.

Penelitian yang dilakukan oleh Pallavi Asrodia dan Hemlata Patel (2012) menunjukkan bahwa Wireshark dapat digunakan untuk memantau data – data yang akan dianalisa dalam penelitian ini, seperti ICMP, dan TCP untuk HTTP GET menggunakan metode yang disebut dengan *packet sniffing*. Pelitian yang dilakukan S.Pavithirakini *et al.* (2016) menyatakan bahwa Wireshark sering digunakan untuk analisis jaringan, salah satunya adalah deteksi *intrusion* jaringan,

atau serangan *Denial of Service* (DoS). 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 dan 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*.
5	Pallavi Asrodia dan Hemlata Patel (2012)	Analysis of Various Packet Sniffing Tools for Network Monitoring and Analysis	Wireshark sebagai perangkat lunak untuk melakukan *packet sniffing*.
6	S.Pavithirakini, *et al.* (2016)	Improve the Capabilities of Wireshark as a tool for Intrusion Detection in DoS Attacks	Wireshark digunakan dalam penelitian *intrusion* jaringan dan serangan DoS

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.

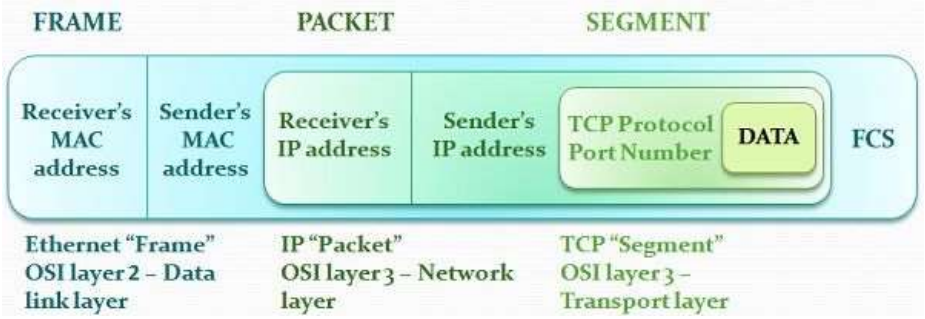
Beberapa contoh rentang alamat IP *bogon* adalah 10.0.0.0/8. Artinya, dalam jaringan ini, jumlah perangkat yang dapat terhubung adalah 16.777.214 dengan rentang IP 10.0.0.1 sampai dengan 10.255.255.254. Contoh rentang alamat IP *bogon* berikutnya adalah 172.16.0.0/12, yang dapat memuat 1.048.574 buah 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 Enkapsulasi alamat MAC, sumber: Techdifference

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. Illustrasi enkapsulasi alamat MAC pada OSI layer kedua seperti pada Gambar 3.1.

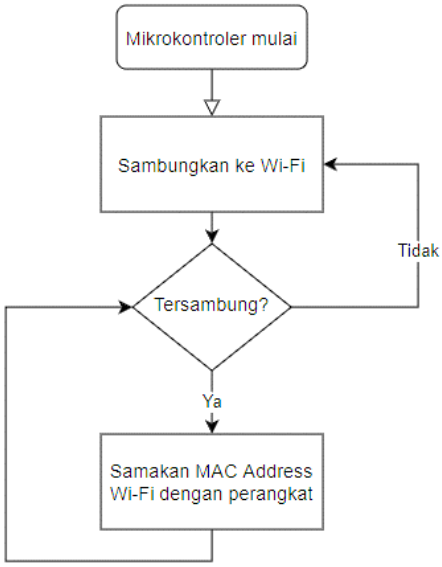
Seiring perkembangannya, *driver, software,* dan sistem operasi terbaru dari NIC memperbolehkan untuk merubah *MAC Address* untuk mencegah *tracking* dari pihak ketiga. Di sisi lain, sifat *hard-coded* dari *MAC Address* tidak berlaku lagi. Kemudahan mengganti *MAC Address* memiliki dampak baik bagi 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.

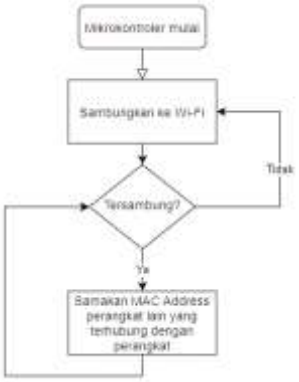
3.3 Penggantian MAC Address

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. Pada VirtualBox, alamat MAC dapat diubah melalui pengaturan alamat MAC pada *Network Card*.



Gambar 3.2 Konfigurasi Pertama

Dengan konfigurasi pada Gambar 3.2, 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 pada Gambar 3.3, 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*. Definisi dari *Time to Live* (TTL) itu sendiri adalah jumlah maksimum dari *hop* yang dapat dilalui paket sebelum diabaikan (*discarded)*. Jumlah maksimum dari TTL didefinisikan oleh perangkat pengirim paket dengan tujuan untuk membatasi jumlah *hop* agar paket tidak berputar (*looping)* pada suatu jaringan dan juga mencegah serangan *Denial-of-Service* (DoS). Sebagai contoh, jika nilai TTL didefinisikan dengan nilai 64, maka paket akan diabaikan pada maksimal router ke-63 sebelum diabaikan. Nilai TTL berkurang 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. Server HTTP menggunakan protokol Transmission Control Protocol (TCP), yang menggunakan proses “three-way handshake”, yang memiliki pola yang mirip seperti *ping request-reply*, namun memiliki fungsionalitas tambahan dengan pola SYN-ACK, yang menandai keberhasilan dikirimnya paket, selain daripada menandai konektivitas *host*.

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)*, seperti pada Gambar 4.1.

```
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, pada Gambar 4.2, 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*

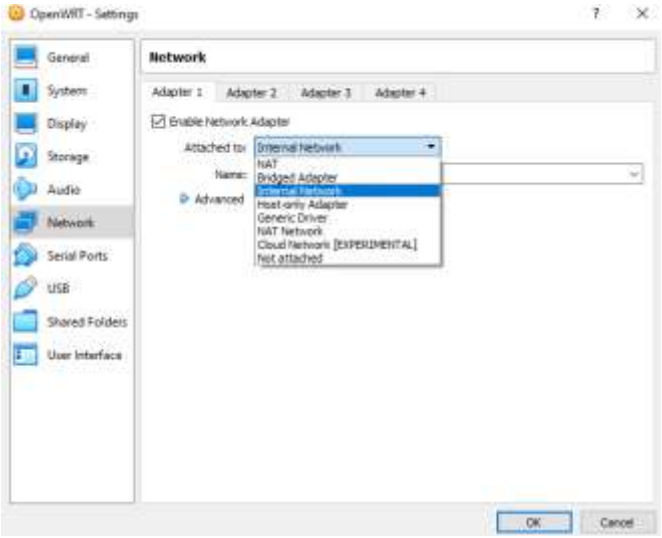
Identifier (OUI) dari Xiaomi Communications Co Ltd. Proses penggantian MAC pada Android dapat dilakukan dengan memasang aplikasi pihak ketiga yang mampu mengubah alamat MAC dengan dukungan akses *root*. Pada penelitian ini, digunakan aplikasi "Change My MAC – Spoof Wifi MAC" yang dikembangkan oleh Banana 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* seperti pada Gambar 4.3, 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

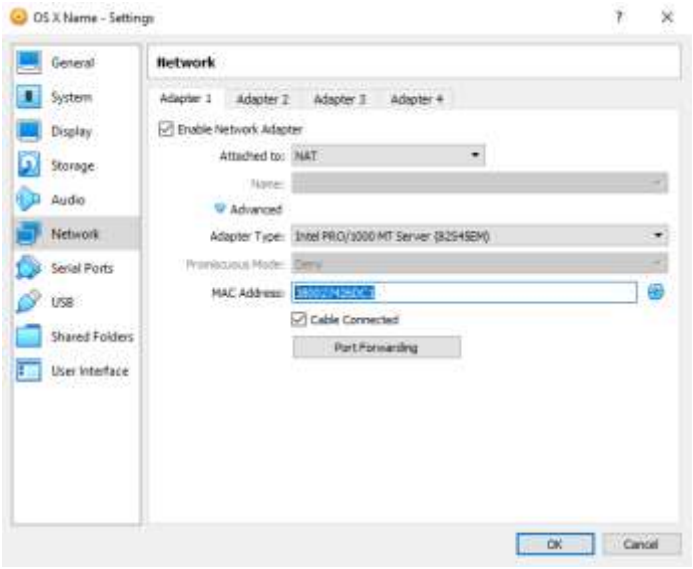
menjalankan VirtualBox tersebut. Mode NAT pada VirtualBox bekerja layaknya sebuah *gateway* atau *router*, dimana *gateway* nya adalah komputer *host* yang menjalankan VirtualBox. NAT pada VirtualBox juga memiliki server *Dynamic Host Configuration Protocol* (DHCP) yang mampu memberikan alamat IP ke beberapa VM yang terhubung secara otomatis.



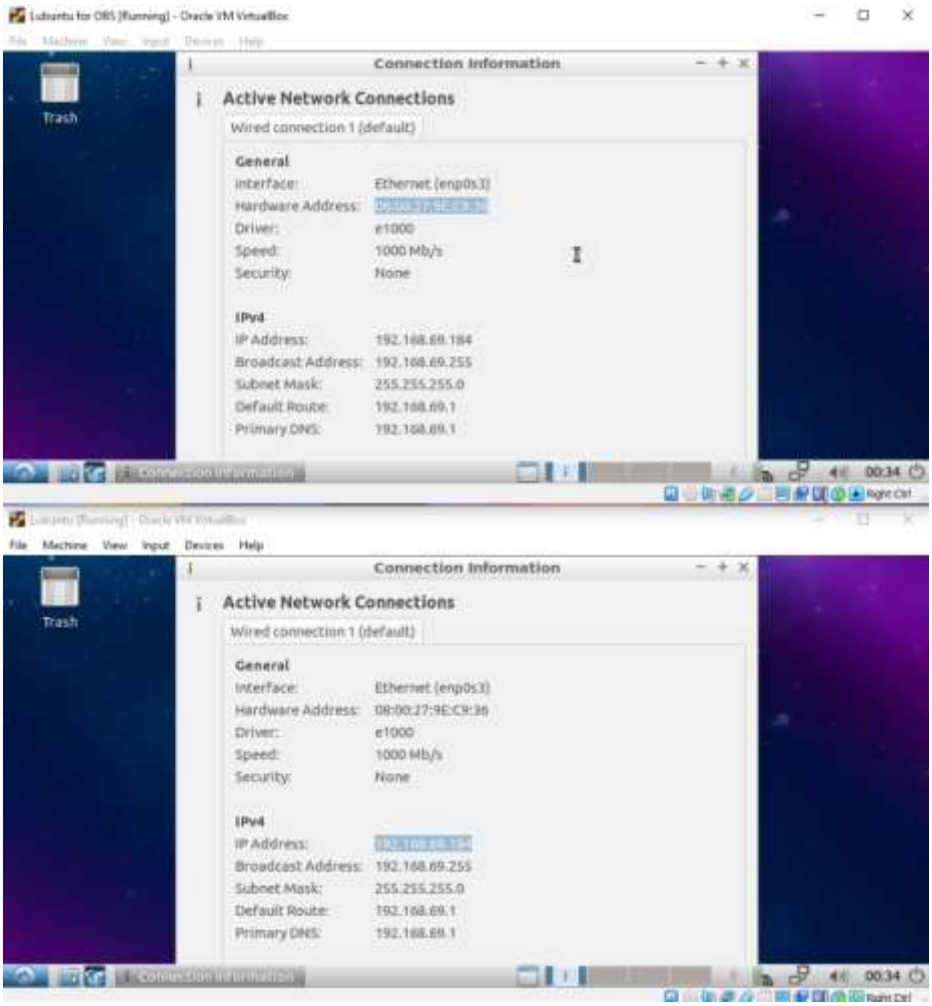
Gambar 4.3 Internal Network

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 , seperti pada Gambar 4.4. 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.

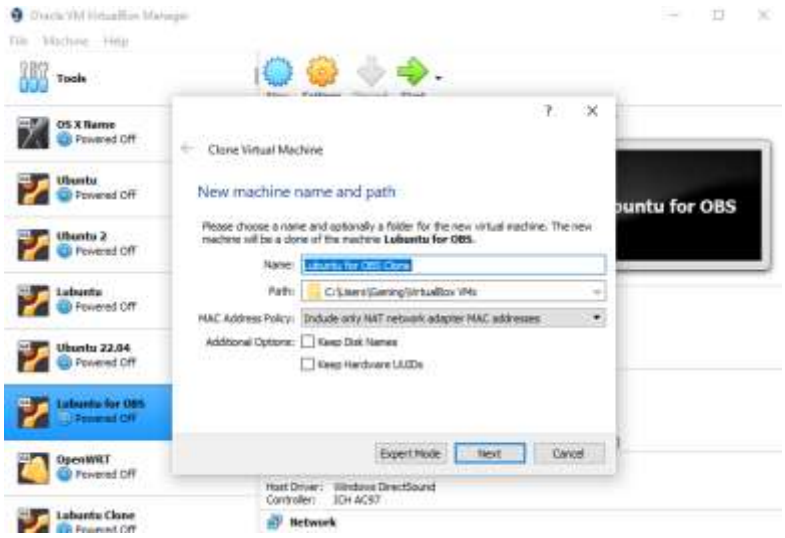


Gambar 4.4 Penggantian MAC Address pada VirtualBox



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* (Gambar 4.6). Dengan demikian, terbentuk dua buah VM identik dengan sistem operasi dan *disk* yang identik.



Gambar 4.6 Clone Virtual Machine

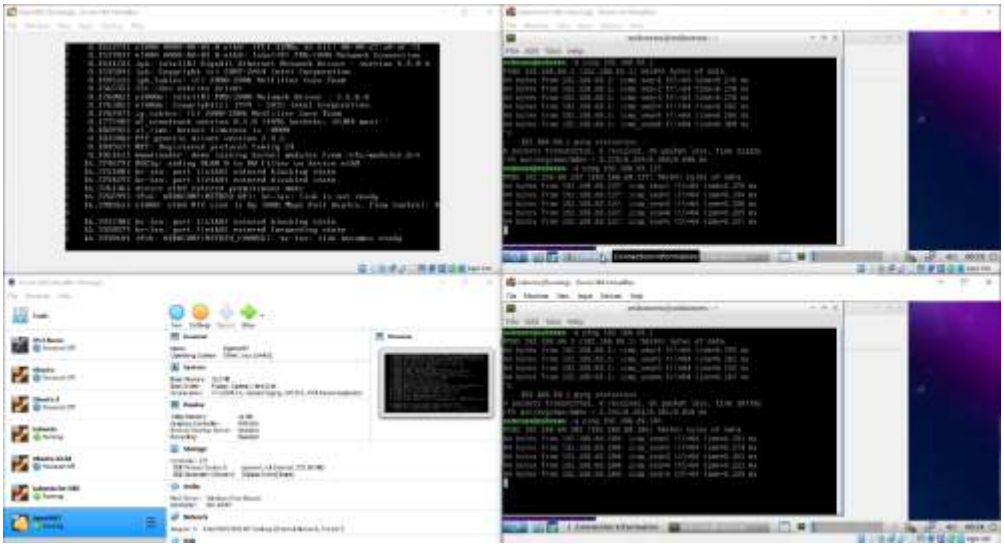
Percobaan dengan *ping* dilakukan dengan menjalankan perintah *ping* (Gambar 4.7) di berbagai OS yang mendukungnya. Pada penelitian ini, semua OS mendukung operasi *ping*. *Ping* dilakukan dengan sintaks *ping*, kemudian diikuti dengan alamat IP atau *hostname*.

```
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

Ping pada sistem operasi Linux Lubuntu dilakukan seperti pada Gambar 4.8, dimana VM OpenWrt berada di pojok kiri atas menjadi tujuan permintaan *ping* dari

kedua VM Linux Lubuntu pada VM yang berada di pojok kanan atas dan pojok kanan bawah.



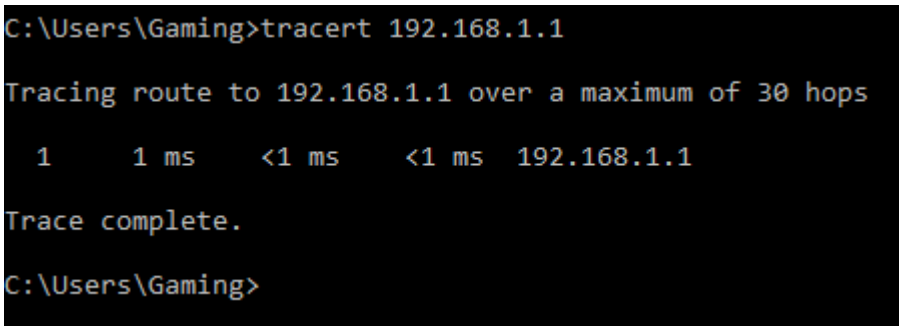
Gambar 4.8 Ping pada dua buah VM

Variasi ping rata – rata atau *average* akan digunakan sebagai variabel terikat dalam penelitian ini, dengan pertimbangan standar deviasi untuk mengamati varians data. 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 sistem operasi Windows (Gambar 4.9), *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

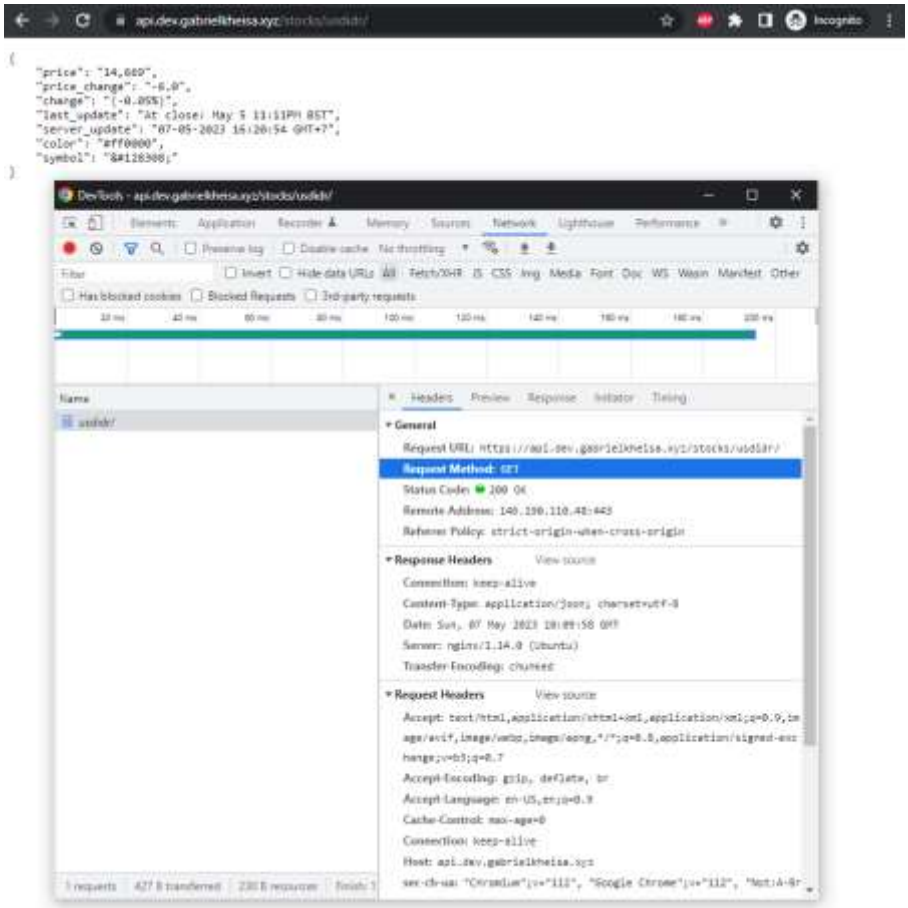
Traceroute menuju *router* (Gambar 4.10) yang melewati satu buah *hop* atau *router* dilakukan pada Gambar 4.10. Percobaan dengan *traceroute* dilakukan dengan menjalankan perintah *traceroute* pada berbagai OS yang mendukungnya. Pada Windows, *traceroute* dapat dilakukan dengan cara mengetik “tracert” pada *command prompt*, diikuti dengan hostname atau IP yang ingin dituju.



Gambar 4.10 Traceroute menuju gateway

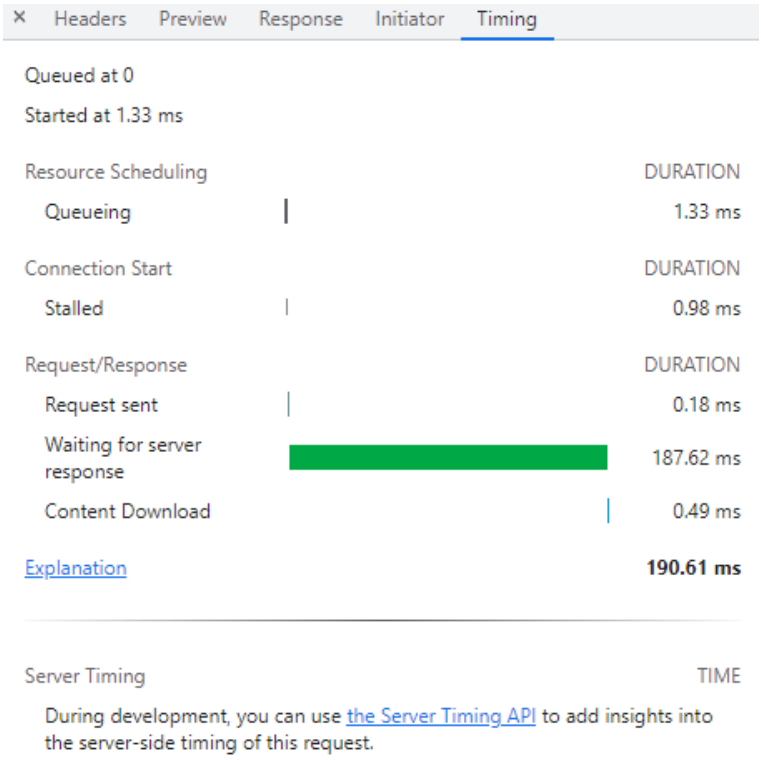
Meskipun *traceroute* hanya merupakan utilitas untuk melakukan pelacakan rute paket, data hasil *traceroute* digunakan sebagai variabel terikat pembanding antara kondisi jaringan dalam keadaan ideal dengan kondisi jaringan dalam keadaan dimana terjadi MAC *address collision*.

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*, seperti pada Gambar 4.11. Hal ini setara dengan HTTP GET yang dilakukan oleh cURL GET.



Gambar 4.11 HTTP GET pada browser Google Chrome

Sebagai tambahan, informasi pewaktuan atau *timing waterfall* juga dapat dilihat pada *developer option*, seperti pada Gambar 4.12.



Gambar 4.12 Informasi timing waterfall

Pada Windows, cURL dapat dilakukan dengan cara mengetik “curl” pada *command prompt*, seperti pada Gambar 4.13, diikuti dengan *flag* “-v” untuk menampilkan statistik dan informasi tambahan pada saat cURL dijalankan, kemudian diikuti dengan hostname atau IP yang ingin dituju.

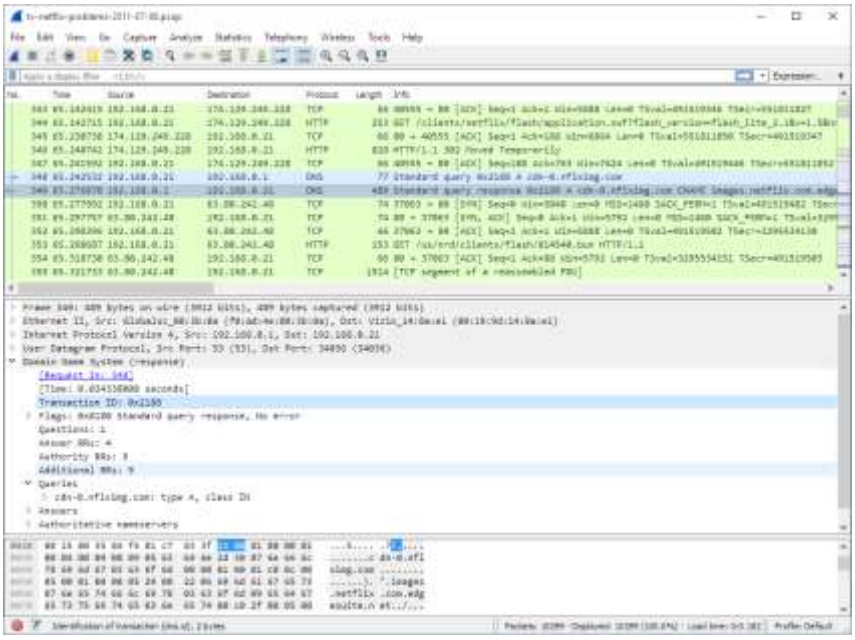
```
C:\Users\Gaming>curl -v https://api.dev.gabrielkheisa.xyz/stocks/usdidr/
*   Trying 146.190.110.48:443...
* Connected to api.dev.gabrielkheisa.xyz (146.190.110.48) port 443 (#0)
* schannel: disabled automatic use of client certificate
* ALPN: offers http/1.1
* ALPN: server accepted http/1.1
> GET /stocks/usdidr/ HTTP/1.1
> Host: api.dev.gabrielkheisa.xyz
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.0 (Ubuntu)
< Date: Sun, 07 May 2023 10:18:30 GMT
< Content-Type: application/json; charset=utf-8
< Transfer-Encoding: chunked
< Connection: keep-alive
<
{
    "price": "14,669",
    "price_change": "-6.0",
    "change": "(-0.05%)",
    "last_update": "At close: May 5 11:11PM BST",
    "server_update": "07-05-2023 16:20:54 GMT+7",
    "color": "#ff0000",
    "symbol": "&#128308;"
}* Connection #0 to host api.dev.gabrielkheisa.xyz left intact
```

Gambar 4.13 HTTP GET melalui cURL

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.

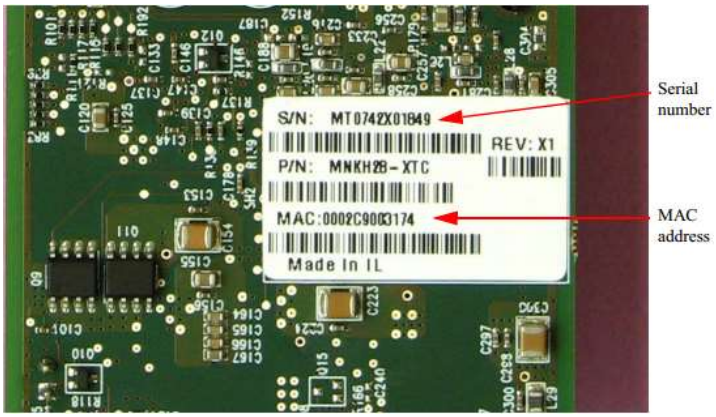
Selanjutnya adalah pengamatan data hasil analisa Wireshark. Wireshark dapat meneliti data yang dikirim melalui jaringan serta protokol – protokol umum seperti protokol jaringan yang terdiri dari TCP, UDP, ICMP, IP, dan ARP, serta meneliti protokol aplikasi, seperti HTTP, FTP, SSH, SMTP, DNS, dan sebagainya. 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. Tangkapan layar untuk Wireshark pada Windows 10 seperti pada Gambar 4.14.



Gambar 4.14 Wireshark Windows 10, sumber: Wireshark

Untuk memperoleh alamat MAC yang akan dijadikan variabel bebas, dapat dilakukan dengan cara mengamati langsung alamat MAC yang tertera pada kartu fisik jaringan, seperti pada Gambar 4.15 dan Gambar 4.16.

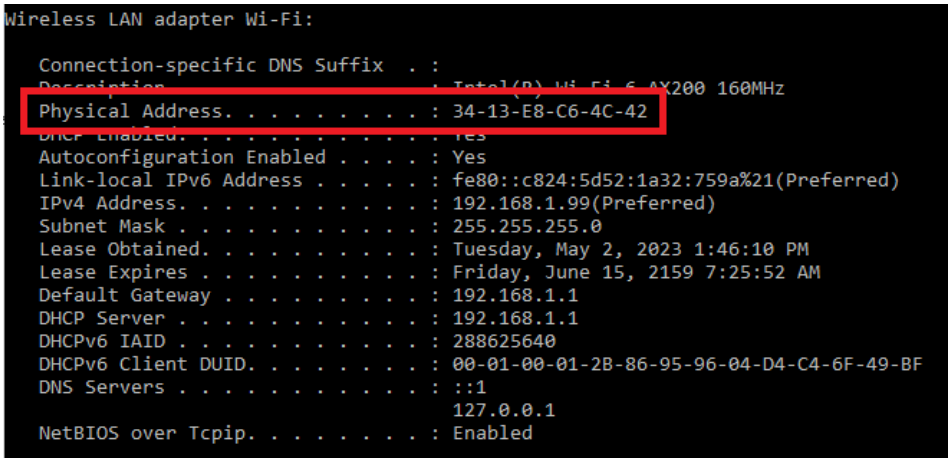
Gambar 4.15 NIC Intel AX200, sumber: Amazon



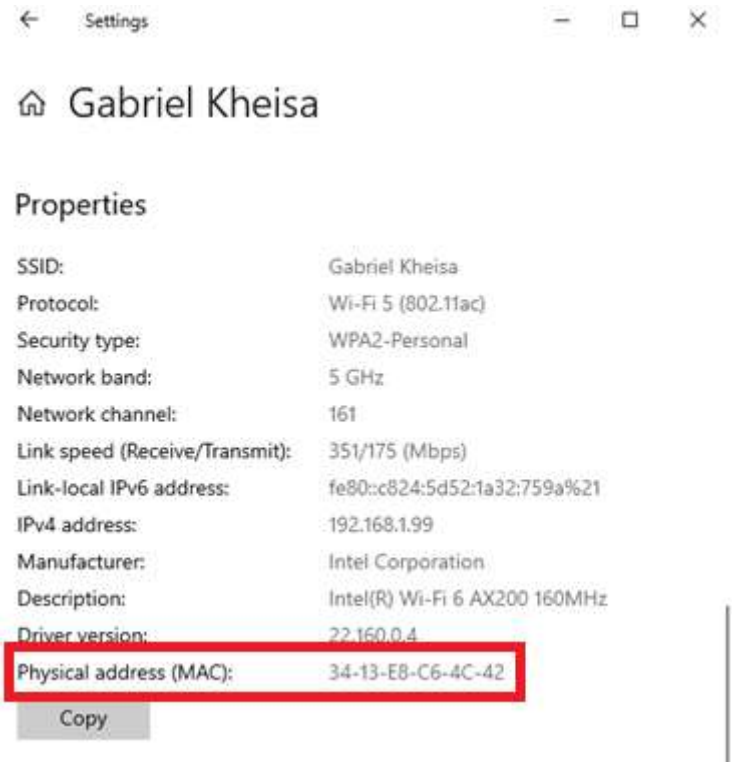
Gambar 4.16 NIC Mellanox, sumber: Nvidia

Namun, alamat MAC juga dapat diperoleh dengan utilitas yang ada pada sistem operasi Windows, seperti *ipconfig* pada *command prompt* (Gambar 4.17), *settings* pada Windows 10 (Gambar 4.18) dan *ifconfig* pada sistem operasi Linux (Gambar 4.19) atau pada pengaturan jaringan pada masing – masing sistem operasi, dan juga

dapat diketahui oleh *administrator* jaringan yang memiliki akses ke *router* (Gambar 4.20).



Gambar 4.17 Alamat MAC pada command prompt



Gambar 4.18 Pengaturan jaringan pada Windows 10

Gambar 4.19 Pengaturan pada Linux Lubuntu

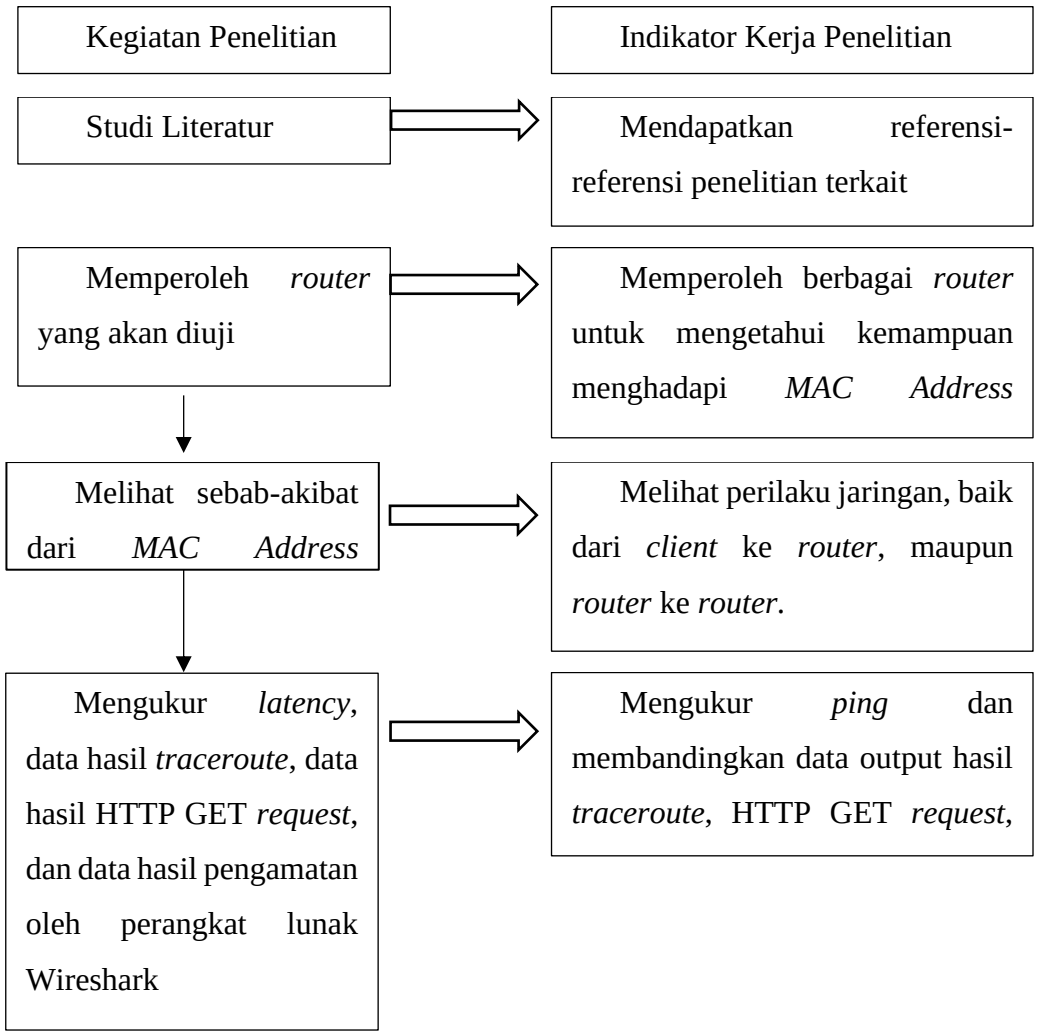
Allocated Address

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 dari tabel DHCP ZTE F609

4.2 Tahapan Penelitian

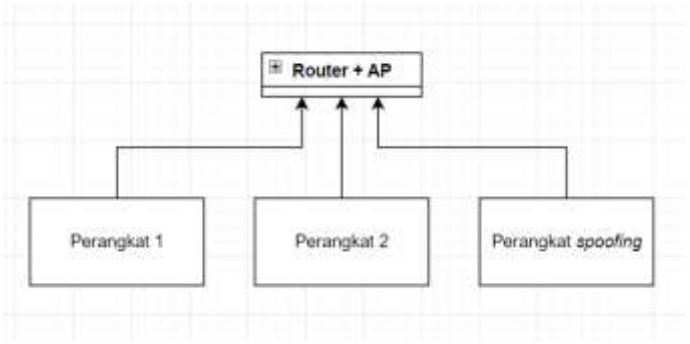
Penelitian skripsi ini memiliki tahapan penelitian yang terbagi menjadi beberapa tahap sebagaimana skema pada Gambar 4.21.



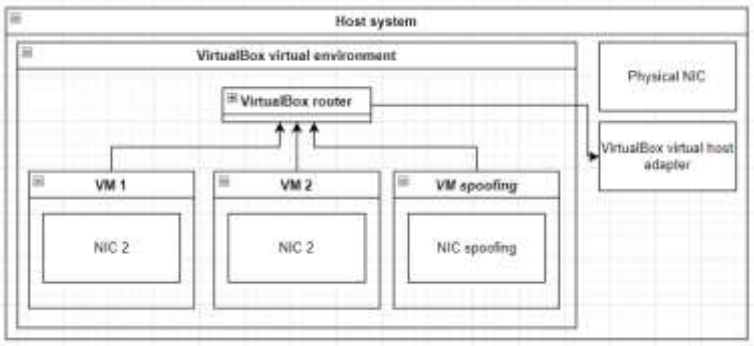
Gambar 4.21 Tahapan Penelitian

4.3 Rancangan sistem

Infrastruktur jaringan untuk pengujian di lingkungan fisik dengan Perangkat 1, Perangkat 2, dan Perangkat yang melakukan *spoofing* alamat MAC seperti pada Gambar 4.22. Kemudian untuk lingkugan virtual dengan VM 1, VM 2, dan VM yang melakukan *spoofing* seperti pada Gambar 4.23.



Gambar 4.22 Topologi pertama (fisik)



Gambar 4.23 Topologi kedua (*virtual environment*)

4.4 Program NodeMCU untuk lingkungan fisik

Pada Gambar 4.24, digunakan *library* yang digunakan untuk menunjang penelitian pada jaringan fisik. ESP8266WiFi.h digunakan untuk mengakses modul Wi-Fi, serta menjalankan fungsi konektivitas jaringan, mengelola koneksi, mengirim data, dan mendeteksi jaringan WiFi. ESP8266Ping.h digunakan untuk menjalankan fungsi *ping*, ESP8266HTTPClient.h untuk menjalankan fungsi permintaan HTTP GET sebagai *client*, WiFiUDP.h digunakan untuk menjalankan

fingsi pengiriman atau penerimaan paket UDP, serta ESP8266WebServer digunakan untuk menjalankan fungsi server HTTP.

```
#include <ESP8266WiFi.h>

#include <ESP8266Ping.h>

#include <ESP8266HTTPClient.h>

#include <WiFiUDP.h>

#include <ESP8266WebServer.h>
```

Gambar 4.24 Library yang digunakan pada NodeMCU

Pada Gambar 4.25, terdapat nilai - nilai MAC address yang akan disamakan. Tipe data merupakan uint8_t, yaitu tipe data integer dengan ukuran 8 bit. Bentuk dari uint8_t adalah uint_8=t newMACAddress[], dimana newMACAddress[] adalah variabel yang menyimpan nilai - nilai MAC address.

Tipe data uint8_t adalah tipe data integer dengan ukuran 8 bit. Nilainya dapat berkisar dari 0 hingga 255. Tipe data ini sering digunakan untuk menyimpan nilai - nilai yang bersifat numerik, seperti alamat IP, port, dan MAC address.

Variabel newMACAddress[] adalah variabel bertipe uint8_t yang memiliki bentuk array. Array adalah tipe data yang dapat menyimpan kumpulan data dengan ukuran tertentu. Dalam hal ini, variabel newMACAddress[] menyimpan 6 nilai uint8_t, yaitu 6 byte dari alamat MAC.

```
// 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 4.25 Data MAC address pada lingkungan fisik

Pada Gambar 4.26, dilakukan 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.

```
1  WiFiClient client;
2
3  ESP8266WebServer server(80);
```

Gambar 4.26 Deklarasi object client dan server

Pada Gambar 4.27, digunakan fungsi ping dari library ping pada ESP8266 untuk mengukur waktu tunda (*latency*) antara ESP8266 dengan *host target*. Fungsi do_ping ini mengirimkan paket ICMP Echo Request ke *host target* dan menunggu paket ICMP *Echo Reply* dari *host target*. Waktu tunda diukur dari waktu pengiriman paket ICMP Echo Request hingga diterimanya paket ICMP Echo Reply dan hasil dari averageTime() merupakan rata – rata nya.

```
1  void do_ping(uint8_t a, uint8_t b, uint8_t c, uint8_t d) {
2    IPAddress remoteIP(a, b, c, d); // Target
3
4    Serial.print("Pinging ");
5    Serial.println(remoteIP);
6
7    int pingTime = Ping.ping(remoteIP);
8    pingTime = Ping.averageTime();
9
10   if (pingTime > 0) {
11     Serial.print("Ping successful! Latency: ");
12     Serial.print(pingTime);
13     Serial.println(" ms");
14   } else {
15     Serial.println("Ping failed!");
16   }
17 }
```

Gambar 4.27 deklarasi fungsi untuk melakukan ping ke alamat IP yang dituju

Pada Gambar 4.28, dibuat fungsi do_curl(url), yang dibuat berdasarkan library HTTPClient dari ESP8266 untuk memeastikan keberhasilan HTTP GET.

```
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 4.28 Deklarasi fungsi curl untuk melakukan HTTP GET

Pada Gambar 4.29, dibuat fungsi http_server() untuk pengujian dimana NodeMCU bertindak sebagai server web. NodeMCU akan mengirimkan pesan "Hello World" kepada *client* yang menghubunginya menggunakan HTTP GET.

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

```

Gambar 4.29 Pesan "Hello, world!" dari HTTP web server.

Pada Gambar 4.30 merupakan deklarasi awal atau void_setup() dari standar Arduino pada NodeMCU. Berikut adalah seluruh deklarasi awal dari seluruh fungsi, objek, dan variabelnya.

```
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 4.30 Deklarasi setup pada Arduino IDE

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(), sesuai pada Gambar 4.31.

```
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 4.31 Deklarasi loop pada Arduino IDE

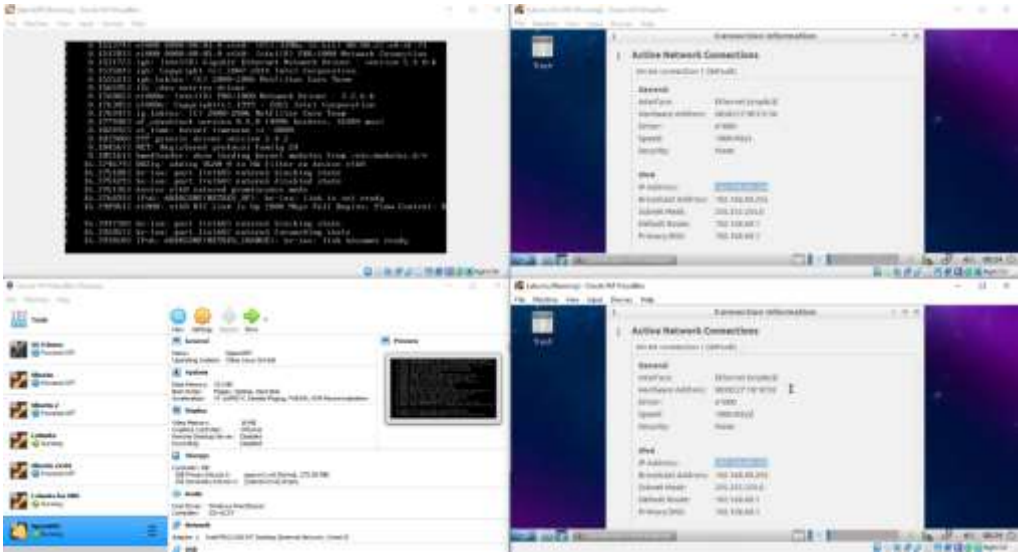
BAB V
HASIL DAN PEMBAHASAN

5.1 Percobaan pada lingkungan virtual dengan VirtualBox

Pada bab ini, dilakukan percobaan pada lingkungan virtual dengan perangkat lunak pendukung VirtualBox.

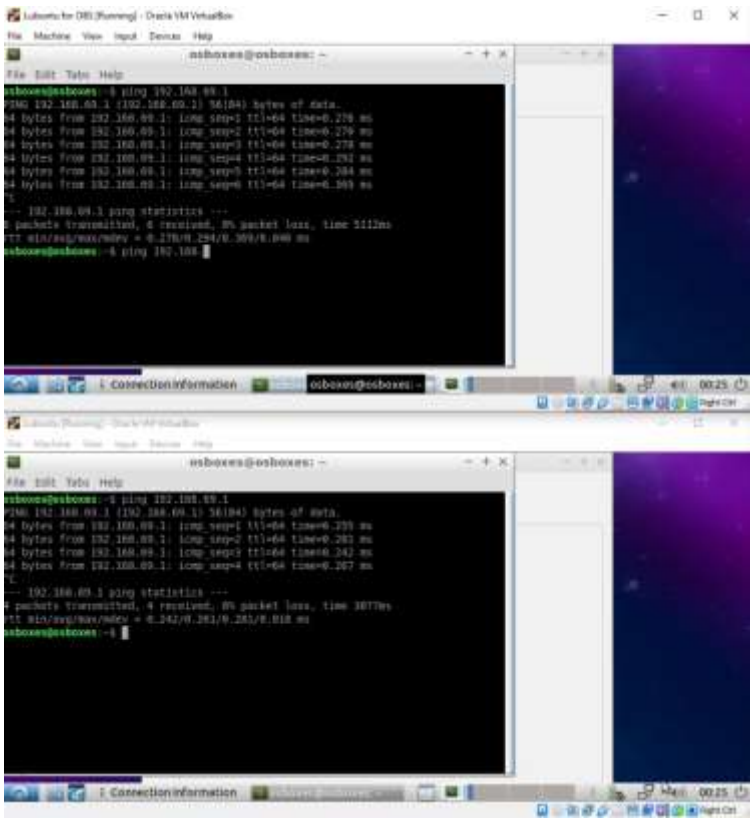
5.1.1 Keadaan Normal (tidak ada duplikasi alamat MAC)

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, seperti pada Gambar 5.1, dimana server OpenWrt berada pada pojok kiri atas, dan kedua VM berada di pojok kanan atas dan pojok kanan bawah.



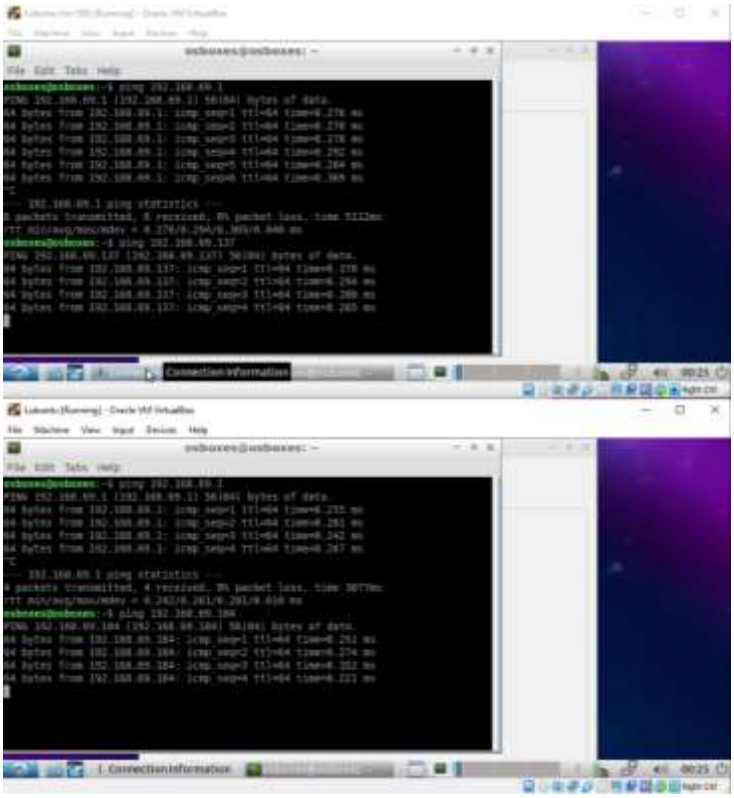
Gambar 5.1 Pengujian pada VirtualBox pada keadaan normal

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, seperti pada Gambar 5.2.



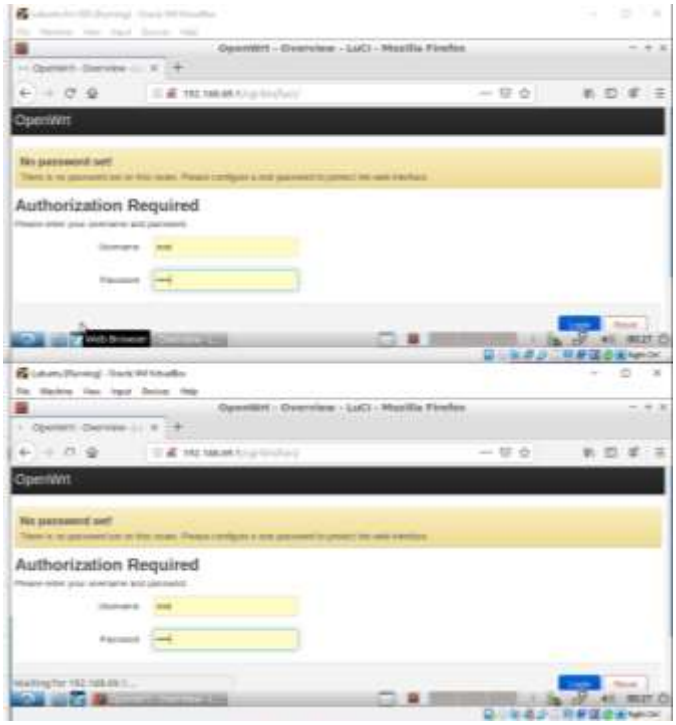
Gambar 5.2 Ping ke gateway

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, seperti pada Gambar 5.3.



Gambar 5.3 Ping ke masing – masing VM satu sama lain

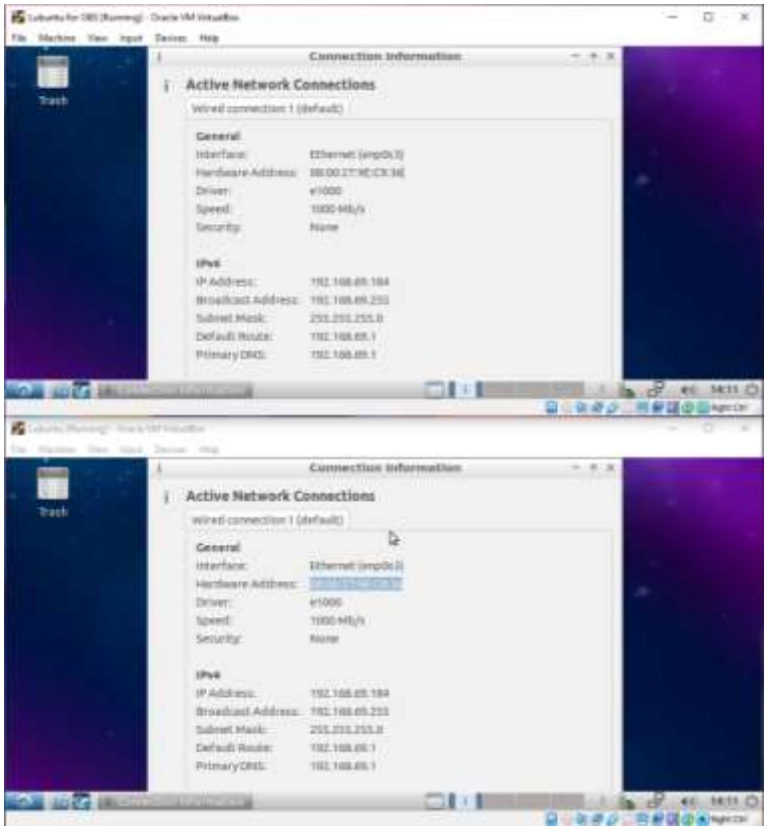
Kemudian, dilakukan pengujian HTTP GET ke VM OpenWRT dengan menggunakan *browser* Mozilla Firefox yang ada pada sistem operasi Lubuntu. Dengan memasukkan alamat *host* atau IP *gateway*, maka hal tersebut setara dengan HTTP GET dengan menggunakan cURL. Hasilnya adalah login page *gateway* dapat diperoleh dengan baik, seperti pada Gambar 5.4.



Gambar 5.4 HTTP GET request menuju login page OpenWRT

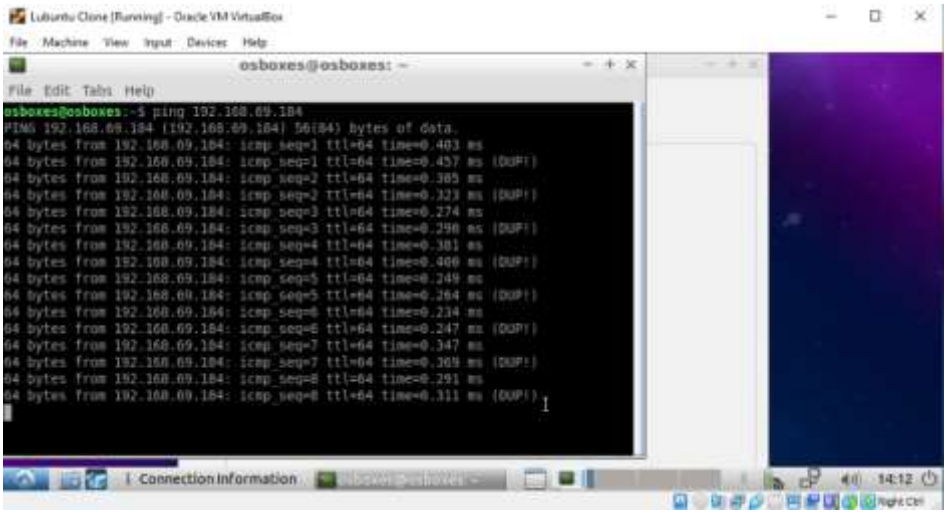
5.1.2 Keadaan dua VM dengan alamat MAC sama dan satu VM pengamat

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 (Gambar 5.5 pojok kanan atas dan pojok kanan bawah). Hanya satu VM yang memiliki satu alamat IP yang unik, yaitu 192.168.69.220, seperti pada Gambar 5.5.



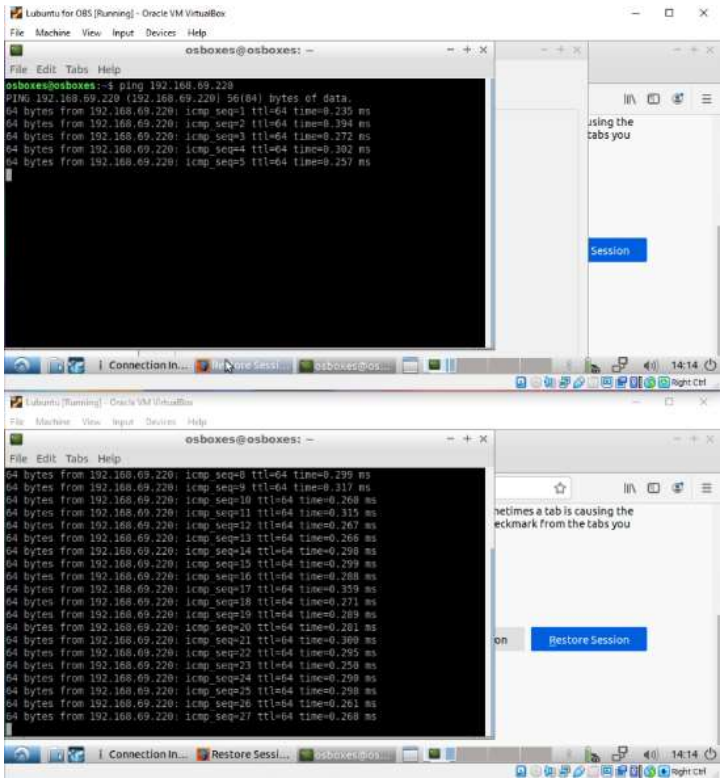
Gambar 5.5 Kedua buah VM memiliki alamat MAC yang sama

Pada saat VM pengamat melakukan *ping* ke alamat IP VM dengan alamat MAC yang sama, hasilnya adalah terjadi respon ganda dengan pesan "(DUP!)", seperti pada Gambar 5.6.



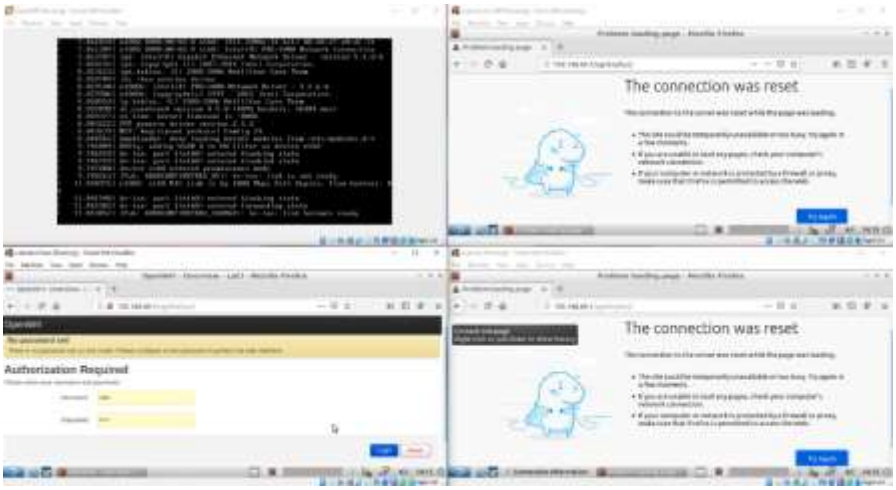
Gambar 5.6 Respon "(DUP!)"

Kemudian, dilakukan *ping* dari masing – masing VM dengan alamat MAC *duplicate* ke VM pengamat. Hasilnya, *ping* tampak normal dan *latency* keduanya berkisar kurang lebih 0.2ms, seperti pada Gambar 5.7.



Gambar 5.7 Ping dari masing – masing VM

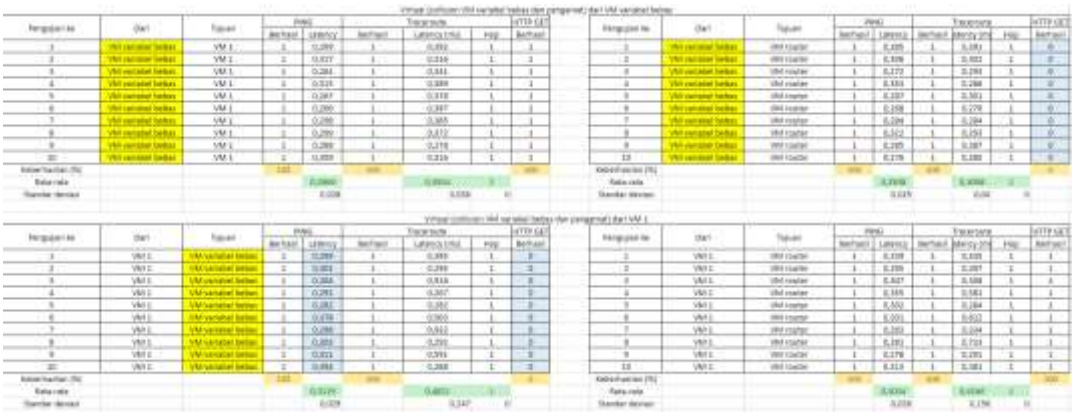
Selanjutnya, dilakukan percobaan HTTP GET ke *webpage* OpenWrt dengan alamat 192.168.69.1 dari masing – masing VM yang terdiri dari dua VM dengan alamat MAC yang sama dan satu VM pengamat. Hasilnya adalah hanya VM pengamat yang dapat menampilkan *webpage* OpenWrt, sedangkan kedua VM *duplicate* tidak dapat menampilkannya, serta terdapat pesan "*The Connection was reset*", seperti pada Gambar 5.8.



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

Hasil masing – masing 10 kali percobaan ditampilkan pada Tabel 5.2 dimana perangkat dengan alamat MAC *duplicate* ditandai dengan baris berwarna kuning, kegagalan HTTP GET ditandai dengan baris berwarna biru pada kolom HTTP GET, serta peningkatan *latency* dan respon "DUP!" ditandai pada kolom *latency* pada *ping* dengan baris berwarna biru.

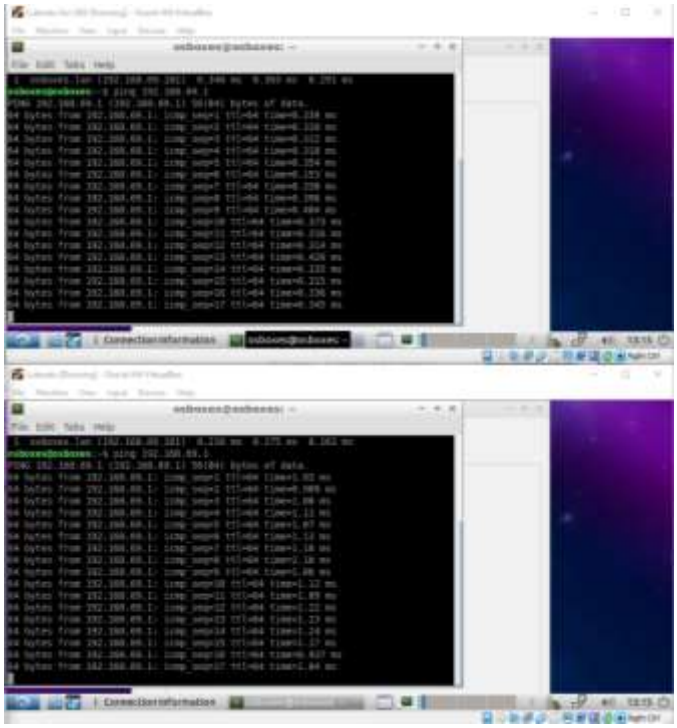
Tabel 5.2 Perbandingan tabel hasil penelitian



5.1.3 Keadaan VM **duplicate** dengan VM OpenWRT

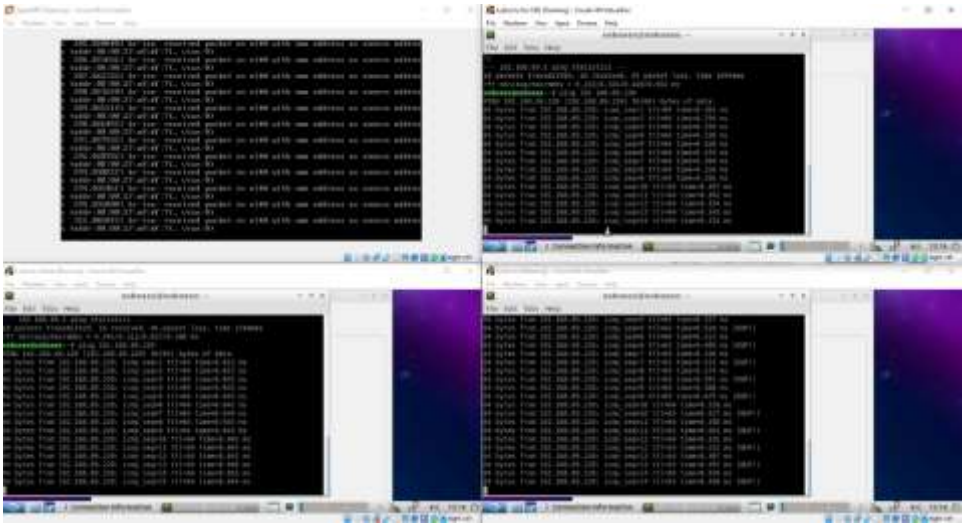
Percobaan berikutnya dengan VirtualBox dalam keadaan salah satu alamat MAC VM memiliki alamat yang sama dengan alamat MAC *router* OpenWRT. Alamat MAC VM yang akan disamakan dengan alamat MAC OpenWRT adalah 08:00:27:D8:98:53, diganti menjadi alamat MAC OpenWRT dengan nilai 08:00:27:A0:DF:71.

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, seperti pada Gambar 5.9.



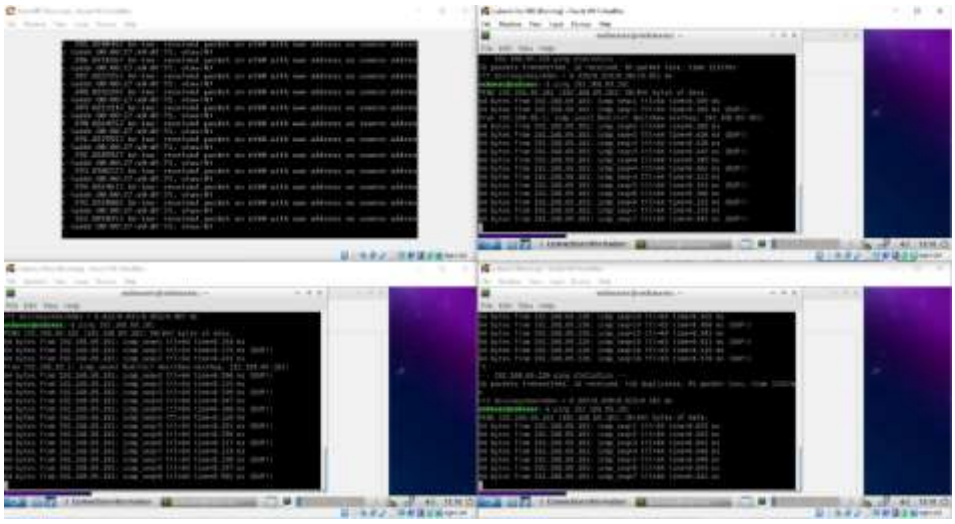
Gambar 5.9 Pengujian ping menuju router OpenWRT

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!”, seperti pada Gambar 5.10.



Gambar 5.10 Pengujian ping menuju VM unik pertama

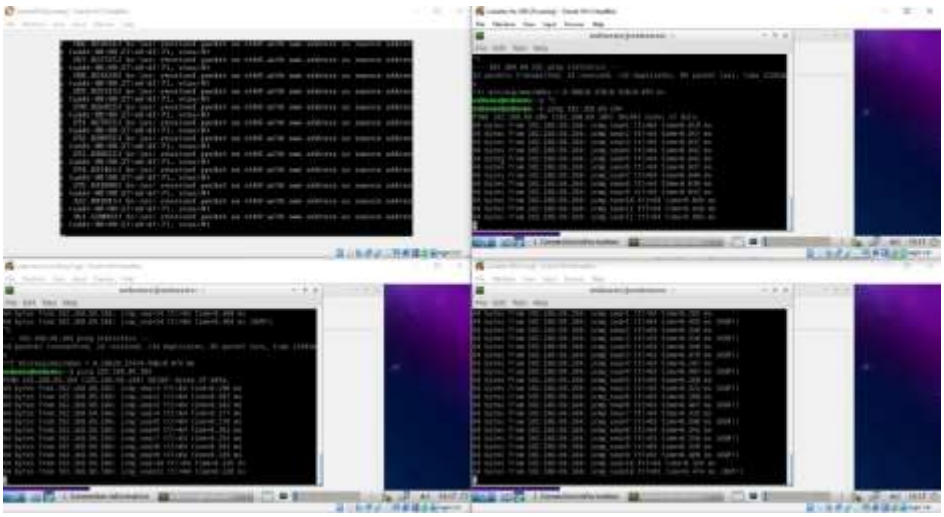
Pengujian *ping* selanjutnya adalah menuju VM dengan alamat MAC *duplicate* yang sama dengan alamat MAC OpenWRT dengan alamat IP 192.168.69.181. Kedua VM dengan alamat MAC yang unik berhasil melakukan *ping* dengan *latency* dibawah 1ms namun terdapat respon ganda "DUP!" dan terdapat pesan "*From 192.168.69.1: icmp_seq=2 Redirect Host(New nexthop: 192.168.69.181)*" pada paket ICMP dengan nomor *sequence* 2, seperti pada Gambar 5.11.



Gambar 5.11 Pengujian ping menuju VM duplicate

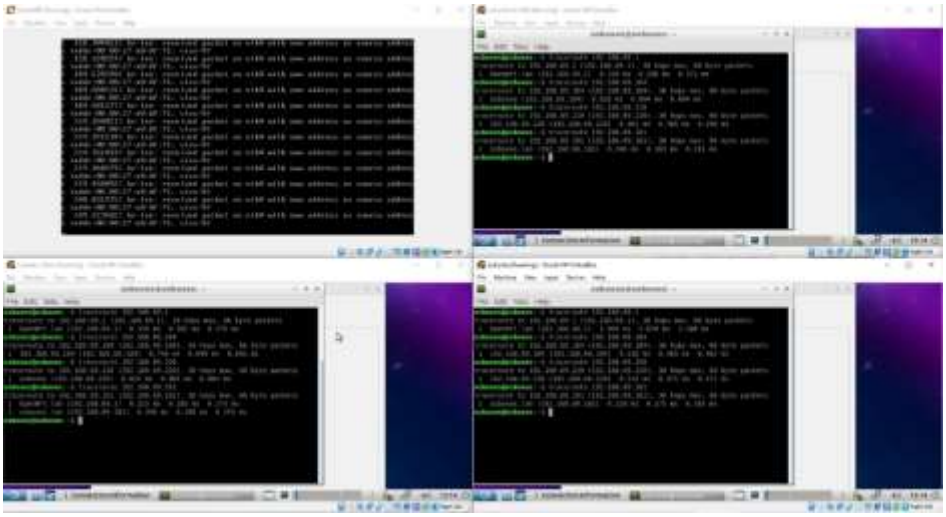
Pengujian *ping* yang terakhir adalah menuju VM dengan alamat MAC unik kedua dengan alamat IP 192.168.69.184. Sama seperti pada percobaan *ping* pada

VM pertama, semua perangkat berhasil melakukan *ping*, namun pada VM *duplicate* terdapat respon ganda dengan pesan “DUP!”, seperti pada Gambar 5.12.



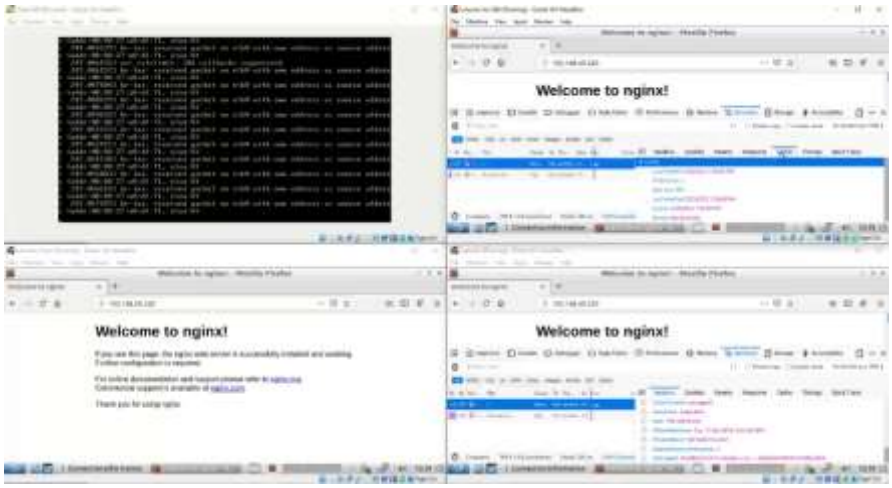
Gambar 5.12 Pengujian ping menuju VM unik kedua

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. Seperti pada Gambar 5.13.

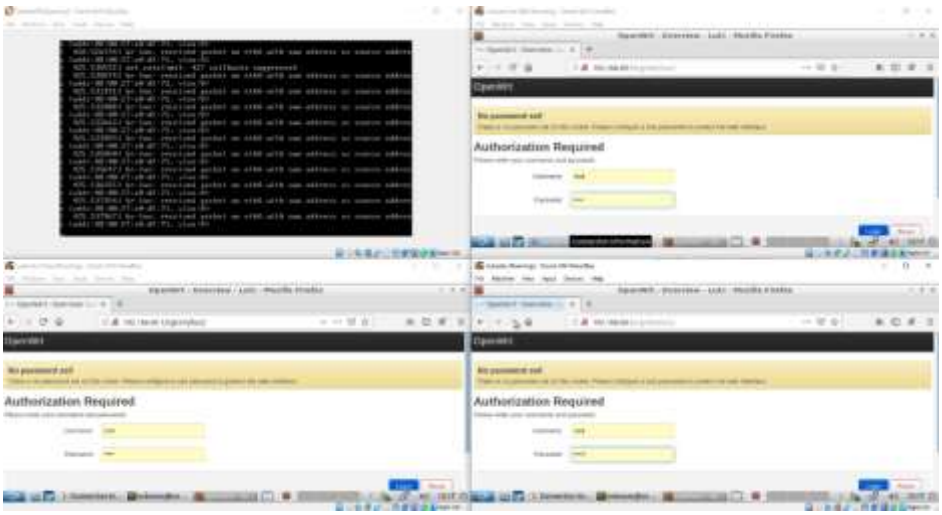


Gambar 5.13 Pengujian traceroute pada masing – masing VM

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, seperti pada Gambar 5.14 dan Gambar 5.15. Tabel percobaan dapat dilihat pada Tabel 5.3.



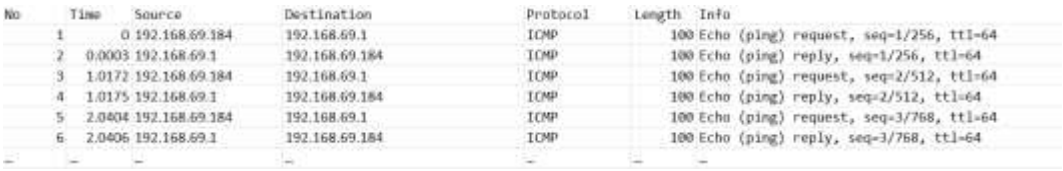
Gambar 5.14 Pengujian HTTP GET dengan server web Nginx



Gambar 5.15 Pengujian HTTP GET menuju webpage OpenWRT

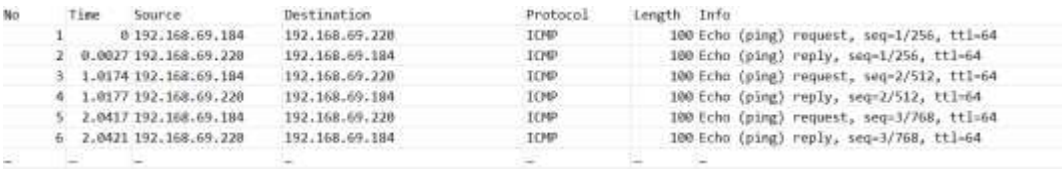
Tabel 5.3 Hasil 10 kali percobaan pada MAC collision VM - Router

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, seperti pada Gambar 5.16 dan Gambar 5.17.



No	Time	Source	Destination	Protocol	Length	Info
1	0	192.168.69.184	192.168.69.1	ICMP	100	Echo (ping) request, seq=1/256, ttl=64
2	0.0003	192.168.69.1	192.168.69.184	ICMP	100	Echo (ping) reply, seq=1/256, ttl=64
3	1.0172	192.168.69.184	192.168.69.1	ICMP	100	Echo (ping) request, seq=2/512, ttl=64
4	1.0175	192.168.69.1	192.168.69.184	ICMP	100	Echo (ping) reply, seq=2/512, ttl=64
5	2.0404	192.168.69.184	192.168.69.1	ICMP	100	Echo (ping) request, seq=3/768, ttl=64
6	2.0406	192.168.69.1	192.168.69.184	ICMP	100	Echo (ping) reply, seq=3/768, ttl=64

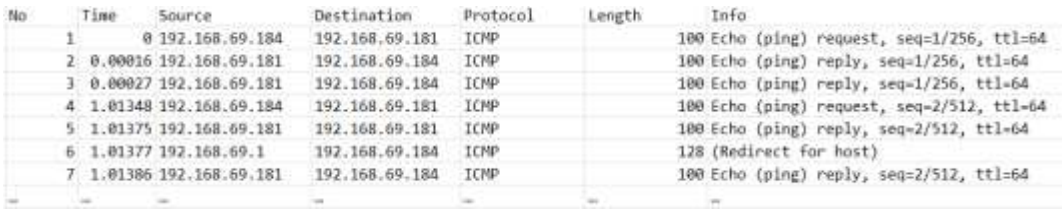
Gambar 5.16 Pengujian ping dari VM pengamat ke VM OpenWRT



No	Time	Source	Destination	Protocol	Length	Info
1	0	192.168.69.184	192.168.69.220	ICMP	100	Echo (ping) request, seq=1/256, ttl=64
2	0.0027	192.168.69.220	192.168.69.184	ICMP	100	Echo (ping) reply, seq=1/256, ttl=64
3	1.0174	192.168.69.184	192.168.69.220	ICMP	100	Echo (ping) request, seq=2/512, ttl=64
4	1.0177	192.168.69.220	192.168.69.184	ICMP	100	Echo (ping) reply, seq=2/512, ttl=64
5	2.0417	192.168.69.184	192.168.69.220	ICMP	100	Echo (ping) request, seq=3/768, ttl=64
6	2.0421	192.168.69.220	192.168.69.184	ICMP	100	Echo (ping) reply, seq=3/768, ttl=64

Gambar 5.17 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*, seperti pada Gambar 5.18.



No	Time	Source	Destination	Protocol	Length	Info
1	0	192.168.69.184	192.168.69.181	ICMP	100	Echo (ping) request, seq=1/256, ttl=64
2	0.00016	192.168.69.181	192.168.69.184	ICMP	100	Echo (ping) reply, seq=1/256, ttl=64
3	0.00027	192.168.69.181	192.168.69.184	ICMP	100	Echo (ping) reply, seq=1/256, ttl=64
4	1.01348	192.168.69.184	192.168.69.181	ICMP	100	Echo (ping) request, seq=2/512, ttl=64
5	1.01375	192.168.69.181	192.168.69.181	ICMP	100	Echo (ping) reply, seq=2/512, ttl=64
6	1.01377	192.168.69.1	192.168.69.184	ICMP	128	(Redirect for host)
7	1.01386	192.168.69.181	192.168.69.184	ICMP	100	Echo (ping) reply, seq=2/512, ttl=64

Gambar 5.18 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. *"Time-to-live exceeded"* merupakan pesan *error* dari paket data yang telah melalui *hop* yang terlalu banyak. TTL akan berkurang setiap kali melalui *hop* dan akan mencapai nilai 0, dimana paket akan diabaikan, seperti pada Gambar 5.19 dan Gambar 5.20.

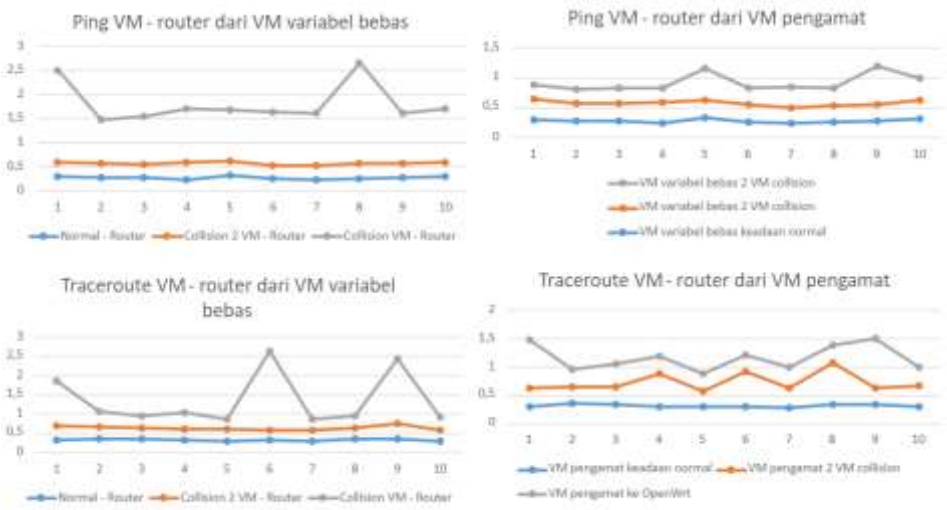
No	Time	Source	Destination	Protocol	Length	Info
1	0	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
2	0.000009	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
3	0.000011	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
4	0.000013	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
5	0.000014	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
6	0.000015	192.168.69.1	192.168.69.184	ICMP	104	Time-to-live exceeded
7	0.000017	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
8	0.000021	192.168.69.1	192.168.69.184	ICMP	104	Time-to-live exceeded
9	0.000022	192.168.69.1	192.168.69.184	ICMP	104	Time-to-live exceeded

Gambar 5.19 Pengujian traceroute dari VM pengamat ke VM duplicate

No	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
2	0.000009	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
3	0.000011	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
4	0.000013	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
5	0.000014	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
6	0.000015	192.168.69.1	192.168.69.184	ICMP	104	Time-to-live exceeded
7	0.000017	192.168.69.181	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
8	0.000021	192.168.69.1	192.168.69.184	ICMP	104	Time-to-live exceeded
9	0.000022	192.168.69.1	192.168.69.184	ICMP	104	Time-to-live exceeded
10	0.000000	192.168.69.1	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
11	0.000022	192.168.69.1	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
12	0.000024	192.168.69.1	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
13	0.000025	192.168.69.1	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
14	0.000071	192.168.69.1	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)
15	0.000075	192.168.69.1	192.168.69.184	ICMP	104	Destination unreachable (Port unreachable)

Gambar 5.20 Pengujian traceroute dari VM pengamat ke VM OpenWRT

Dari 10 kali percobaan berikut, didapat bahwa *latency* pada keadaan normal memiliki *latency* dengan rata – rata terkecil dibandingan *latency* pada keadaan *collision* saat melakukan *ping* dan *traceroute* ke *router*. *Latency* juga lebih beragam (dilihat dari standar deviasi dan grafik) pada beberapa skenario *collision*, terutama *collision* alamat MAC pada VM dan *router* pada statistik Gambar 5.21.



Gambar 5.21 Statistik latency MAC address collision pada lingkungan virtual

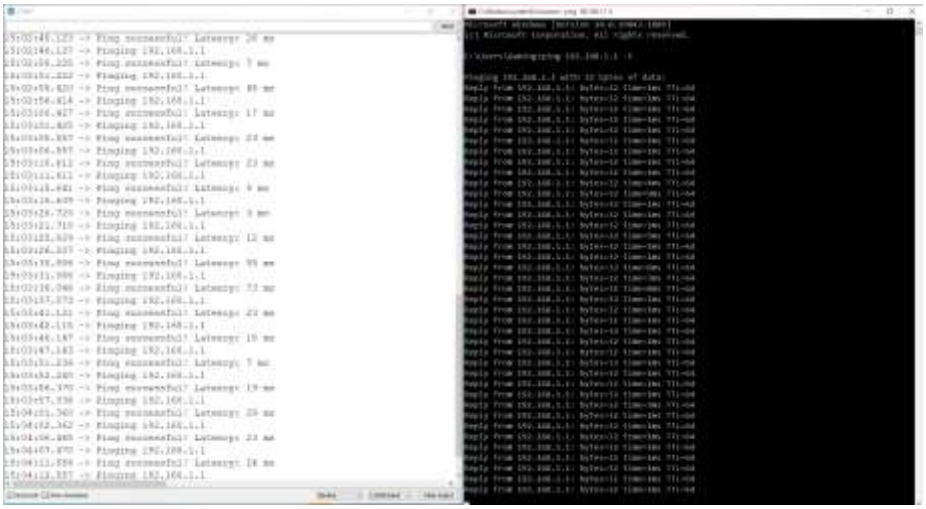
5.2 Percobaan pada lingkungan fisik

Pada bab ini, dilakukan percobaan pada lingkungan fisik, dengan perangkat keras berupa NodeMCU sebagai perangkat variabel bebas, dimana perangkat ini bertindak sebagai perangkat yang dapat mengubah alamat MAC, serta Komputer dan Android sebagai client..

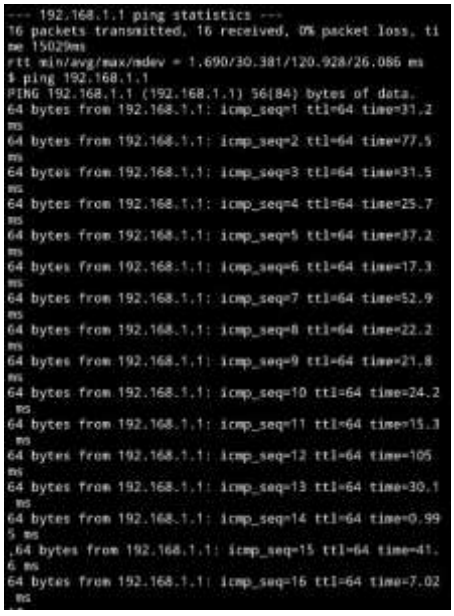
5.2.1 Keadaan Normal (tidak ada duplikasi alamat MAC)

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, seperti pada Gambar 5.22 dan Gambar 5.23.

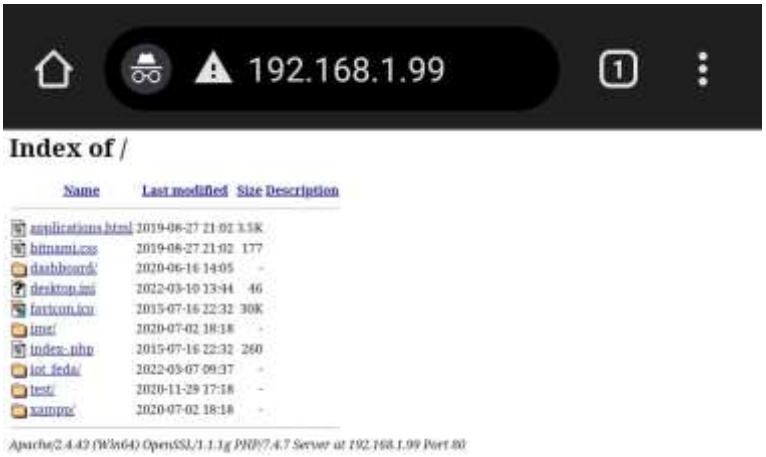


Gambar 5.22 Ping NodeMCU (kiri) dan ping komputer (kanan) menuju router



Gambar 5.23 Ping Android menuju router

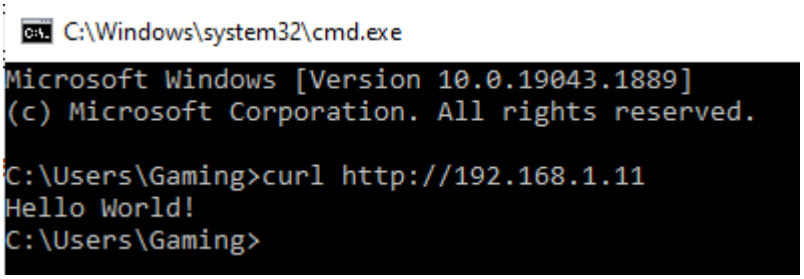
Pengujian GET dilakukan dengan menjalankan server Apache pada komputer dan pada *homepage router*. Baik server Apache dan *homepage router* dapat menampilkan halaman server, baik melalui cURL maupun aplikasi *browser*, seperti pada Gambar 5.24, Gambar 5.25, Gambar 5.26 dan Gambar 5.27.



Gambar 5.24 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 5.25 HTTP GET dari NodeMCU menuju router dan komputer



Gambar 5.26 HTTP GET dari komputer ke NodeMCU

Gambar 5.27 HTTP GET dari komputer ke Android

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*.

Pada Tabel 5.7, data *latency* juga lebih bervariatif dan memiliki nilai rata – rata diatas rata – rata dibandingkan pada lingkungan *virtual*.

Tabel 5.4 Hasil keadaan normal

Fisik (normal) ke router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	1	26	1
2	NodeMCU	Router	1	7	1
3	NodeMCU	Router	1	45	1
4	NodeMCU	Router	1	17	1
5	NodeMCU	Router	1	23	1
6	NodeMCU	Router	1	23	1
7	NodeMCU	Router	1	9	1
8	NodeMCU	Router	1	3	1
9	NodeMCU	Router	1	12	1
10	NodeMCU	Router	1	56	1
Keberhasilan (%)			100		100
Rata-rata				22.1	
Standar deviasi				16.0714	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	1	31.2	1
2	Android	Router	1	77.5	1
3	Android	Router	1	31.5	1
4	Android	Router	1	25.7	1
5	Android	Router	1	37.2	1
6	Android	Router	1	17.3	1
7	Android	Router	1	52.9	1
8	Android	Router	1	22.2	1
9	Android	Router	1	21.8	1
10	Android	Router	1	24.2	1
Keberhasilan (%)			100		100
Rata-rata				34.15	
Standar deviasi				17.33628	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	1	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	1	1
4	Komputer	Router	1	1	1
5	Komputer	Router	1	1	1
6	Komputer	Router	1	1	1
7	Komputer	Router	1	1	1
8	Komputer	Router	1	1	1
9	Komputer	Router	1	1	1
10	Komputer	Router	1	4	1
Keberhasilan (%)			100		100
Rata-rata				1.3	
Standar deviasi				0.9	

Fisik (normal) ke komputer

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	1	20	1
2	NodeMCU	Komputer	1	26	1
3	NodeMCU	Komputer	1	13	1
4	NodeMCU	Komputer	1	27	1
5	NodeMCU	Komputer	1	17	1
6	NodeMCU	Komputer	1	20	1
7	NodeMCU	Komputer	1	19	1
8	NodeMCU	Komputer	1	42	1
9	NodeMCU	Komputer	1	47	1
10	NodeMCU	Komputer	1	11	1
Keberhasilan (%)			100		100
Rata-rata				24.2	
Standar deviasi				11.2321	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Komputer	1	304	1
2	Android	Komputer	1	25.5	1
3	Android	Komputer	1	77.6	1
4	Android	Komputer	1	281.5	1
5	Android	Komputer	1	35.2	1
6	Android	Komputer	1	136.8	1
7	Android	Komputer	1	97.2	1
8	Android	Komputer	1	234.4	1
9	Android	Komputer	1	243.2	1
10	Android	Komputer	1	48.9	1
Keberhasilan (%)			100		100
Rata-rata				148.43	
Standar deviasi				101.9301	

Fisik (normal) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	1	248	1
2	NodeMCU	Android	1	364	1
3	NodeMCU	Android	1	202	1
4	NodeMCU	Android	1	278	1
5	NodeMCU	Android	1	396	1
6	NodeMCU	Android	1	182	1
7	NodeMCU	Android	1	269	1
8	NodeMCU	Android	1	265	1
9	NodeMCU	Android	1	358	1
10	NodeMCU	Android	1	92	1
Keberhasilan (%)			100		100
Rata-rata				265.4	
Standar deviasi				87.67577	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	1	316	1
2	Komputer	Android	1	301	1
3	Komputer	Android	1	73	1
4	Komputer	Android	1	447	1
5	Komputer	Android	1	351	1
6	Komputer	Android	1	66	1
7	Komputer	Android	1	361	1
8	Komputer	Android	1	46	1
9	Komputer	Android	1	204	1
10	Komputer	Android	1	171	1
Keberhasilan (%)			100		100
Rata-rata				233.6	
Standar deviasi				134.4457	

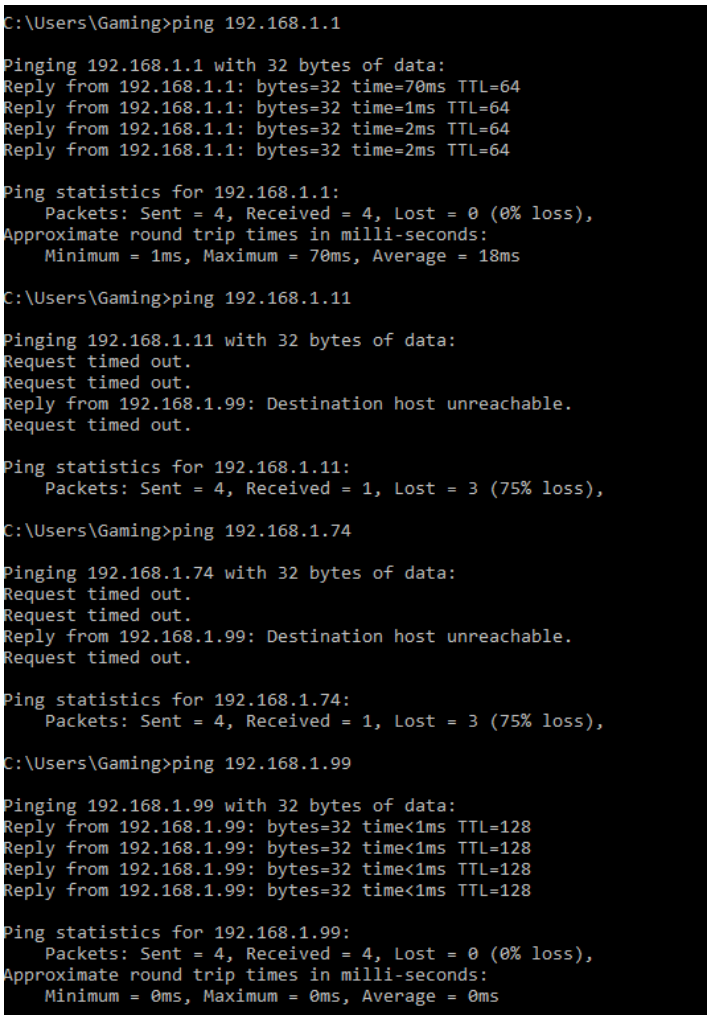
5.2.2 Keadaan MAC NodeMCU *duplicate* dengan *router*

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, seperti pada Gambar 5.28.

```
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
```

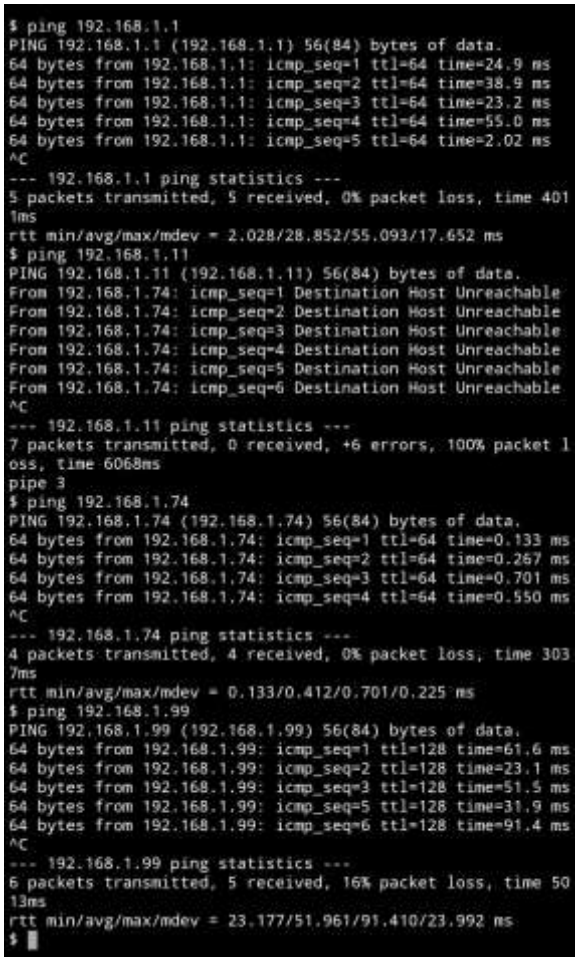
Gambar 5.28 Ping dan HTTP GET dari NodeMCU ke semua perangkat

Sedangkan *ping* dari komputer hanya berhasil menuju *router* dengan *latency* rata – rata yang lebih besar, seperti pada Gambar 5.29.



Gambar 5.29 Ping dari Komputer ke semua perangkat

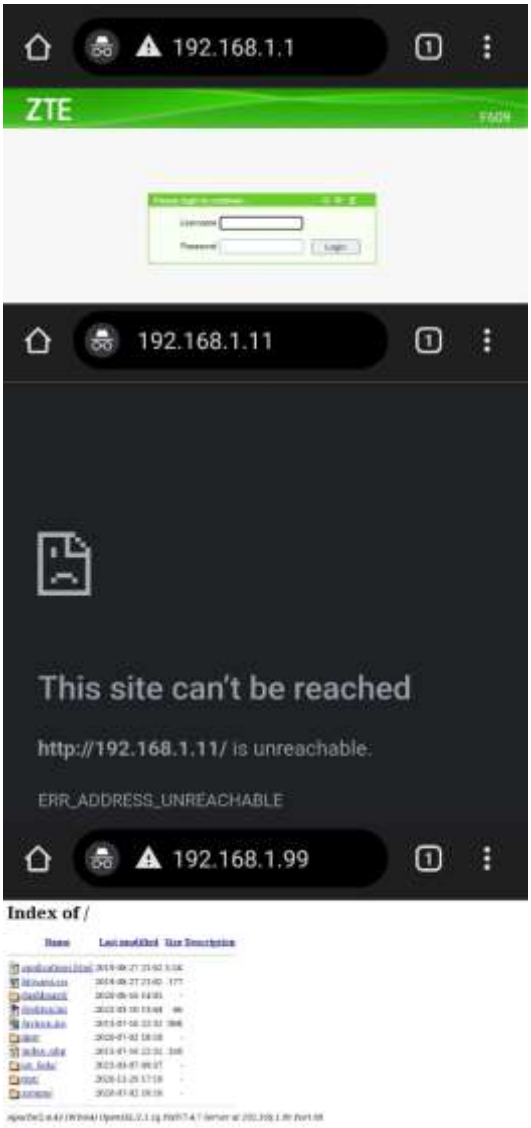
Kemudian *ping* dari Android hanya berhasil menuju *router* dan komputer, seperti pada Gambar 5.30.



Gambar 5.30 Ping dari Android ke semua 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, seperti pada Gambar 5.31 dan Gambar 5.32. Untuk tabel hasil seperti pada Tabel 5.5.

Gambar 5.31 HTTP GET dari komputer ke semua perangkat



Gambar 5.32 HTTP GET dari Android ke semua perangkat

Tabel 5.5 Hasil pengujian lingkungan fisik collision dengan Router

Fisik (NodeMCU router) ke router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	0		0
2	NodeMCU	Router	0		0
3	NodeMCU	Router	0		0
4	NodeMCU	Router	0		0
5	NodeMCU	Router	0		0
6	NodeMCU	Router	0		0
7	NodeMCU	Router	0		0
8	NodeMCU	Router	0		0
9	NodeMCU	Router	0		0
10	NodeMCU	Router	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	1	24	1
2	Android	Router	1	38.9	1
3	Android	Router	1	23.2	1
4	Android	Router	1	55	1
5	Android	Router	1	2.02	1
6	Android	Router	1	34.9	1
7	Android	Router	1	60.3	1
8	Android	Router	1	24.5	1
9	Android	Router	1	55.4	1
10	Android	Router	1	21.7	1
Keberhasilan (%)			100		100
Rata-rata				33.992	
Standar deviasi				17.5965	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	70	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	2	1
4	Komputer	Router	1	2	1
5	Komputer	Router	1	2	1
6	Komputer	Router	1	1	1
7	Komputer	Router	1	2	1
8	Komputer	Router	1	97	1
9	Komputer	Router	1	2	1
10	Komputer	Router	1	2	1
Keberhasilan (%)			100		100
Rata-rata				18.1	
Standar deviasi				33.25492	

Fisik (NodeMCU router) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	0		0
2	NodeMCU	Android	0		0
3	NodeMCU	Android	0		0
4	NodeMCU	Android	0		0
5	NodeMCU	Android	0		0
6	NodeMCU	Android	0		0
7	NodeMCU	Android	0		0
8	NodeMCU	Android	0		0
9	NodeMCU	Android	0		0
10	NodeMCU	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	0		0
2	Komputer	Android	0		0
3	Komputer	Android	0		0
4	Komputer	Android	0		0
5	Komputer	Android	0		0
6	Komputer	Android	0		0
7	Komputer	Android	0		0
8	Komputer	Android	0		0
9	Komputer	Android	0		0
10	Komputer	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Fisik (NodeMCU router) ke Komputer

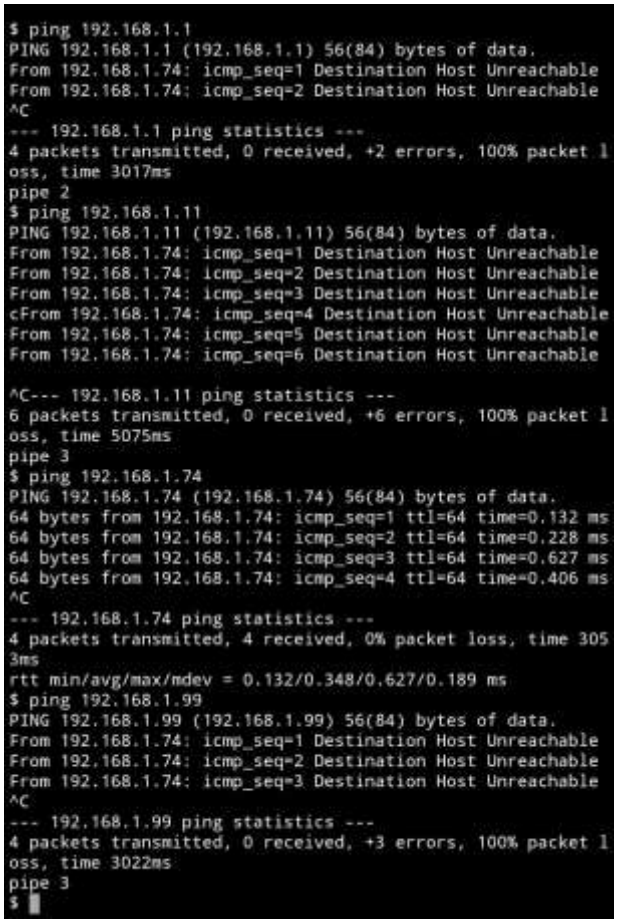
Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	0		0
2	NodeMCU	Komputer	0		0
3	NodeMCU	Komputer	0		0
4	NodeMCU	Komputer	0		0
5	NodeMCU	Komputer	0		0
6	NodeMCU	Komputer	0		0
7	NodeMCU	Komputer	0		0
8	NodeMCU	Komputer	0		0
9	NodeMCU	Komputer	0		0
10	NodeMCU	Komputer	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Komputer	1	61.6	1
2	Android	Komputer	1	23.1	1
3	Android	Komputer	1	51.5	1
4	Android	Komputer	1	31.9	1
5	Android	Komputer	1	91.4	1
6	Android	Komputer	1	75.3	1
7	Android	Komputer	1	62.5	1
8	Android	Komputer	1	87.1	1
9	Android	Komputer	1	19.6	1
10	Android	Komputer	1	76	1
Keberhasilan (%)			100		100
Rata-rata				58	
Standar deviasi				24.56603	

5.2.3 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, seperti pada Gambar 5.33, Gambar 5.34, Gambar 5.35, Gambar 5.36, dan Gambar 5.37.

Gambar 5.33 Ping dari komputer ke semua perangkat



Gambar 5.34 Ping dari Android ke semua perangkat

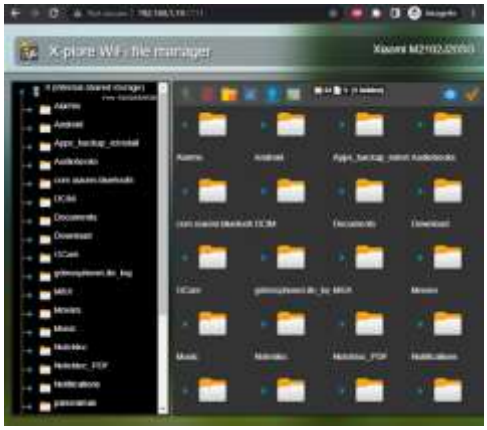
```
21:53:35.371 -> Pinging 192.168.1.11
21:53:40.410 -> Ping failed!
21:53:40.410 -> Pinging 192.168.1.1
21:53:44.457 -> Ping successful! Latency: 59 ms
21:53:44.457 -> Pinging 192.168.1.74
21:53:49.500 -> Ping failed!
21:53:49.500 -> Pinging 192.168.1.99
21:53:53.689 -> Ping successful! Latency: 37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

Gambar 5.35 Ping dari NodeMCU ke semua 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 5.36 HTTP GET dari NodeMCU ke semua perangkat



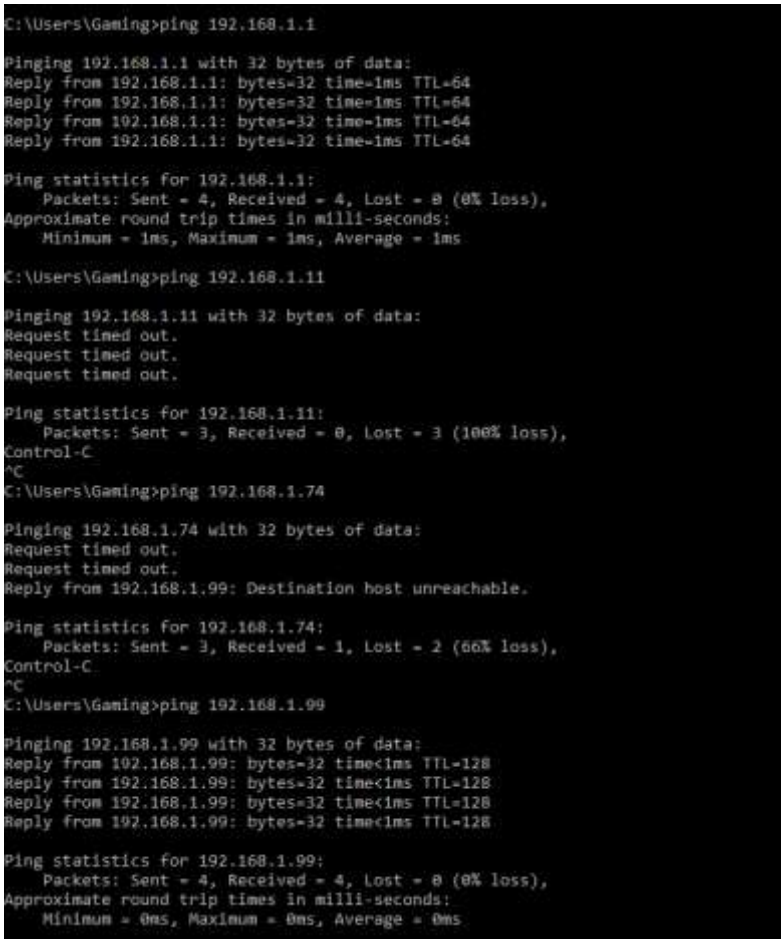
Gambar 5.37 HTTP GET dari komputer ke Android

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

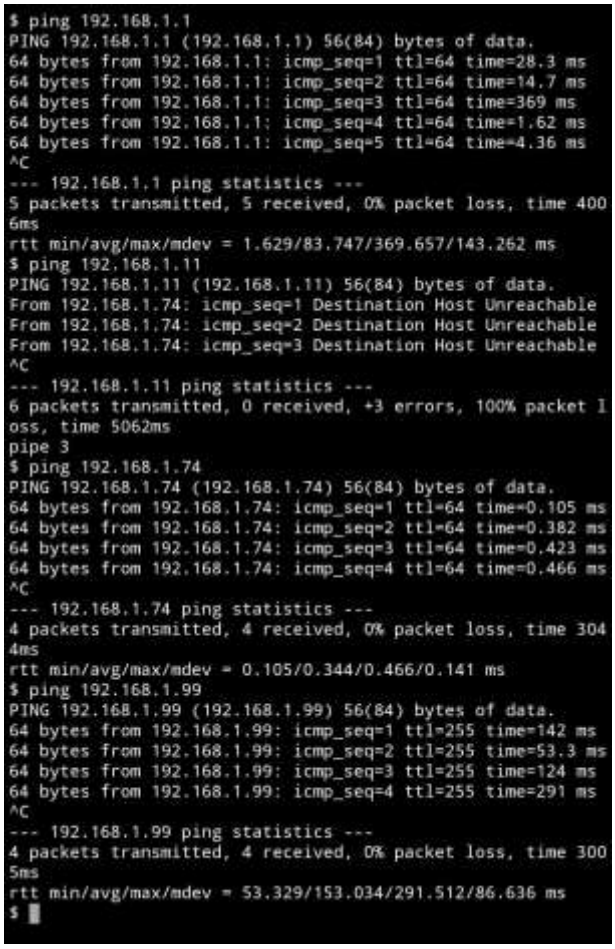
Untuk *ping* dalam keadaan MAC NodeMCU sama dengan perangkat komputer, *ping* menunjukkan RTO pada *ping* dari NodeMCU ke semua perangkat, dari komputer menuju Android. *Ping* perangkat Android menuju *router* dan komputer relatif lebih besar dibandingkan pada keadaan normal, seperti pada Gambar 5.38, Gambar 5.39, dan Gambar 5.40.

```
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 5.38 Ping dan HTTP GET dari NodeMCU ke semua perangkat

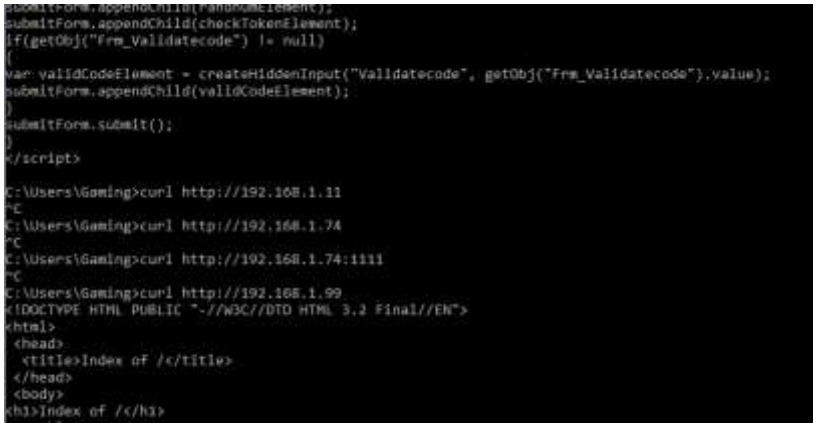


Gambar 5.39 Ping dari komputer ke semua perangkat

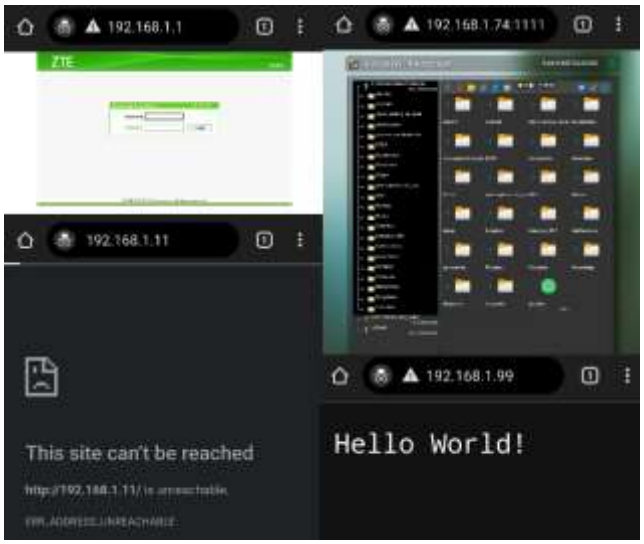


Gambar 5.40 Ping dari Android ke semua perangkat

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, seperti pada Gambar 5.41 dan Gambar 5.42.



Gambar 5.41 HTTP GET komputer ke semua perangkat



Gambar 5.42 HTTP GET Android ke semua perangkat

Ketidaksesuaian respon HTTP GET tersebut ditandai pada Tabel 5.6 pada pengujian HTTP GET dar Android ke Komputer dengan warna biru tua.

Tabel 5.6 Hasil collision NodeMCU – Komputer

Fisik (NodeMCU router) ke Router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	0		1
2	NodeMCU	Router	0		1
3	NodeMCU	Router	0		1
4	NodeMCU	Router	0		1
5	NodeMCU	Router	0		1
6	NodeMCU	Router	0		1
7	NodeMCU	Router	0		1
8	NodeMCU	Router	0		1
9	NodeMCU	Router	0		1
10	NodeMCU	Router	0		1
Keberhasilan (%)			0		100
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	1	28.3	1
2	Android	Router	1	14.7	1
3	Android	Router	1	969	1
4	Android	Router	1	1.62	1
5	Android	Router	1	4.36	1
6	Android	Router	1	61.87	1
7	Android	Router	1	4.72	1
8	Android	Router	1	230	1
9	Android	Router	1	28	1
10	Android	Router	1	59.03	1
Keberhasilan (%)			100		100
Rata-rata				80.16	
Standar deviasi				115.742	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	1	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	1	1
4	Komputer	Router	1	1	1
5	Komputer	Router	1	1	1
6	Komputer	Router	1	3	1
7	Komputer	Router	1	1	1
8	Komputer	Router	1	1	1
9	Komputer	Router	1	1	1
10	Komputer	Router	1	1	1
Keberhasilan (%)			100		100
Rata-rata				1.2	
Standar deviasi				0.6	

Fisik (NodeMCU router) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	0		0
2	NodeMCU	Android	0		0
3	NodeMCU	Android	0		0
4	NodeMCU	Android	0		0
5	NodeMCU	Android	0		0
6	NodeMCU	Android	0		0
7	NodeMCU	Android	0		0
8	NodeMCU	Android	0		0
9	NodeMCU	Android	0		0
10	NodeMCU	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

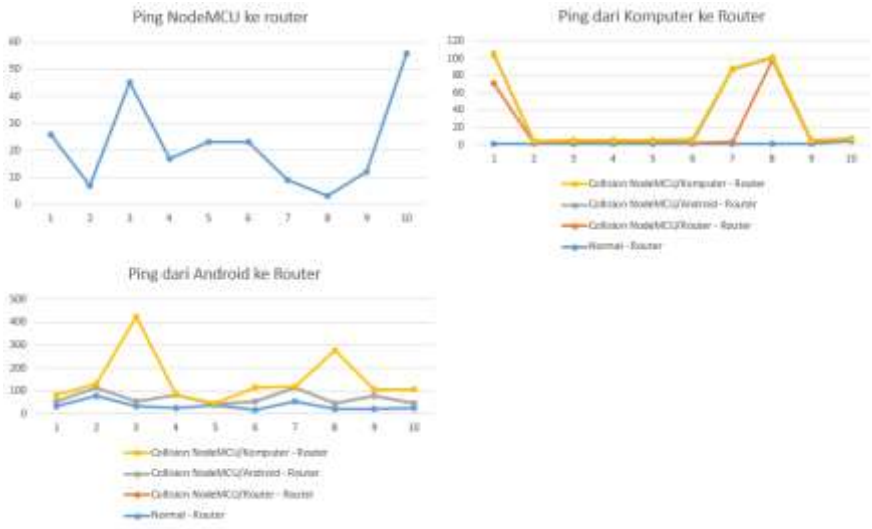
Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	0		0
2	Komputer	Android	0		0
3	Komputer	Android	0		0
4	Komputer	Android	0		0
5	Komputer	Android	0		0
6	Komputer	Android	0		0
7	Komputer	Android	0		0
8	Komputer	Android	0		0
9	Komputer	Android	0		0
10	Komputer	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Fisik (NodeMCU router) ke Komputer

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	0		0
2	NodeMCU	Komputer	0		0
3	NodeMCU	Komputer	0		0
4	NodeMCU	Komputer	0		0
5	NodeMCU	Komputer	0		0
6	NodeMCU	Komputer	0		0
7	NodeMCU	Komputer	0		0
8	NodeMCU	Komputer	0		0
9	NodeMCU	Komputer	0		0
10	NodeMCU	Komputer	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Komputer	1	142	0
2	Android	Komputer	1	53.5	0
3	Android	Komputer	1	124	0
4	Android	Komputer	1	291	0
5	Android	Komputer	1	115.02	0
6	Android	Komputer	1	186.54	0
7	Android	Komputer	1	149.57	0
8	Android	Komputer	1	75.91	0
9	Android	Komputer	1	245.75	0
10	Android	Komputer	1	188.63	0
Keberhasilan (%)			100		0
Rata-rata				157.192	
Standar deviasi				69.37191	

Statistik pada Gambar 5.42 juga menunjukkan bahwa rata – rata *latency* pada keadaan *collision* diatas rata – rata keadaan normal.



Gambar 5.43 Statistik 10 percobaan keadaan normal

BAB VI
KESIMPULAN DAN SARAN

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, namun pada lingkungan fisik perlu diuji kembali dengan perangkat identik.
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 di VirtualBox berhasil menampilkan webpage melalui browser maupun cURL. Namun, di lingkungan fisik, hanya beberapa perangkat yang berhasil, dan terdapat respon HTTP GET yang seharusnya ditampilkan oleh perangkat yang dituju, terutama ketika melakukan HTTP GET dari Android ke komputer. Hasil penelitian juga mencatat peningkatan rata-rata *latency* dari 0,2 milisekon menjadi 1,2 milisekon, baik di lingkungan virtual maupun fisik. Selain itu, terdapat kegagalan konektivitas dalam beberapa skenario percobaan, dan selama 10 kali percobaan di lingkungan fisik, 100% mendapatkan respon HTTP GET yang tidak sesuai saat terjadi MAC *Address Collision.*

6.2 Saran

Adapun saran dari penelitian ini adalah sebagai berikut:

1. Untuk memperkuat hasil penelitian, perlu dilakukan pengujian ulang pada lingkungan fisik dengan perangkat yang identik. Hal ini untuk memastikan bahwa hasil penelitian tidak dipengaruhi oleh perbedaan perangkat.
2. Mengingat nilai latency pada lingkungan fisik lebih beragam, penelitian perlu dilakukan lebih dari 10 kali. Hal ini untuk mendapatkan hasil yang lebih akurat dan dapat digeneralisasi.
3. Untuk mendapatkan hasil yang lebih komprehensif, pengujian pada VirtualBox dapat menggunakan hub, bukan switch. Hal ini memungkinkan VM pengamat untuk melihat payload antara VM kedua, VM ketiga, dan ke VM router.

DAFTAR PUSTAKA

Alasdair Allan. 2017. “List of MAC addresses with vendors identities”. GitHub. https://gist.github.com/aallan/b4bb86db86079509e6159810ae9bd3e4.

Android 6.0 changes. 2015. https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html, 2015.

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

Cisco. 2015. “Configuring Port Security”.

F. Baker. 2015. Baker, F (ed.). “Requirements for IP Version 4 Routers”. p. 52. RFC 1812.

Institute of Electrical and Electronics Engineers. 2004. “Overview and Guide to the IEEE 802 LMSC”. https://grouper.ieee.org/groups/802/802%20overview.pdf.

Jarmo Mölsä. 2006. "Mitigating DoS Attacks against the DNS with Dynamic TTL Values". Networking Laboratory, Helsinki University of Technology. http://lib.tkk.fi/Diss/2006/isbn9512282151/article2.pdf.

Jean-François Determe, Sophia Azzagnuni, François Horlin, Philippe De Doncker. 2020. "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.

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

Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, Frank Piessens. 2016. "Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms". iMinds-Distrinet, KU Leuven, Univ Lyon, INSA Lyon, Inria, CITI, France.

Oracle Corporation. 2022. "Changelog for VirtualBox 6.1". https://www.virtualbox.org/wiki/Changelog-6.1.

Pallavi Asrodia, Hemlata Patel. 2012. "Analysis of Various Packet Sniffing Tools for Network Monitoring and Analysis". Department of Computer Science and Engineering, Jawaharlal Institute of Technology, Borawan, Khargone.

Peng Jiang, Hongyi Wu, Cong Wang, Chunsheng Xin. 2018. "Virtual MAC Spoofing Detection through Deep Learning". Department of ECE, Old Dominion University, Norfolk, VA, USA.

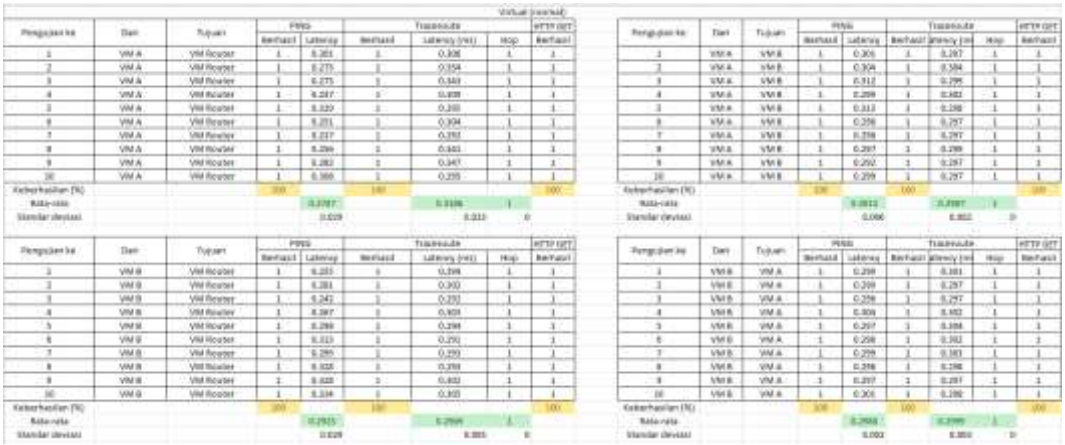
S. Pavithirakini, D.D.M.M. Bandara, C.N. Gunawardhana, K.K.S. Perera, B.G.M.M. Abeyrathne, Dhishan Dhammearatchi. 2016. "Improve the Capabilities of Wireshark as a Tool for Intrusion Detection in DoS Attacks". Sri Lanka Institute of Information Technology Computing Pvt. Ltd.

Tang Yong, Liu Xiaoyan. 2007. "System, Node as Well as Method for Detecting MAC Address Collision in Loop Network". Huawei Technologies Co Ltd.

Xerox Corporation. 1984. "Xerox System Integration Standard 098404 - Authentication Protocol".

LAMPIRAN

Lampiran 1. Virtual – Keadaan Normal

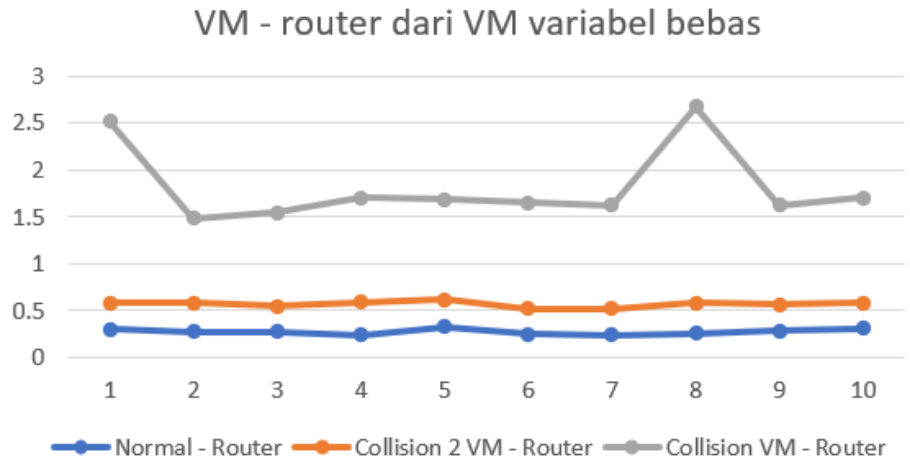


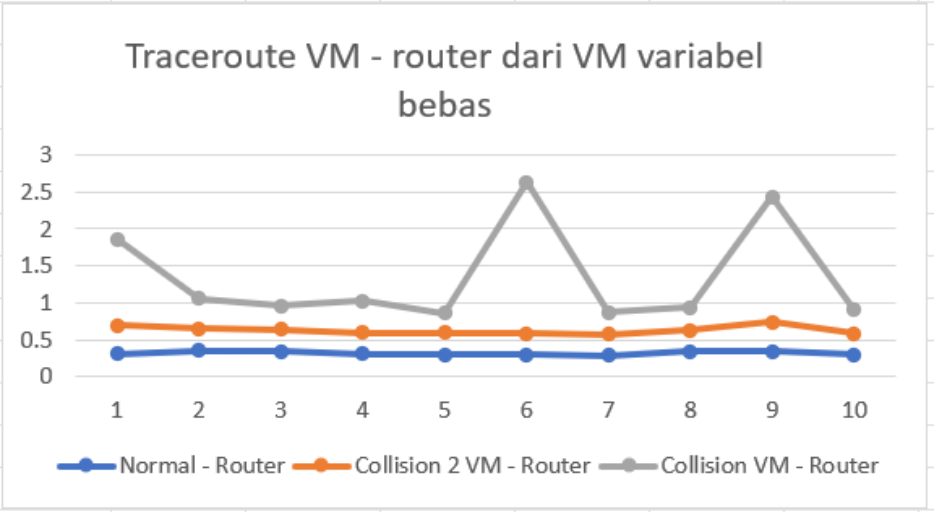
Lampiran 2. Virtual – dua VM dengan alamat sama

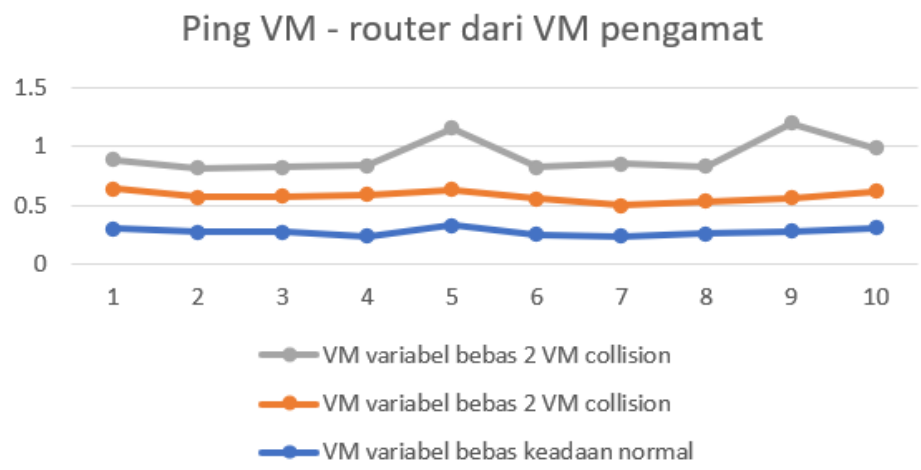
Lampiran 3. Virtual – VM duplicate denggan OpenWrt

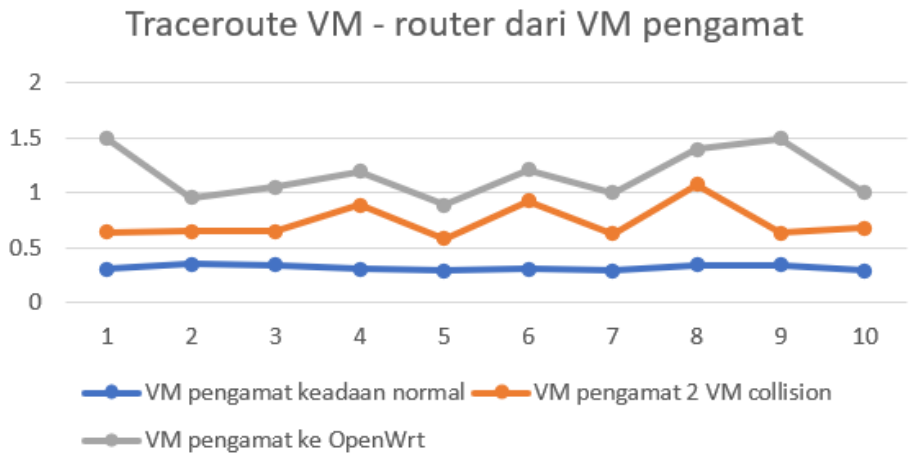
Ping dengan label biru DUP!

Lampiran 4. Virtual – Statistik






Ping VM - router dari VM pengamat
1.5
1
0.5
0
1
2
3
4
5
6
7
8
9
10
VM variabel bebas 2 VM collision
VM variabel bebas 2 VM collision
VM variabel bebas keadaan normal


Traceroute VM - router dari VM pengamat
2
1.5
1
0.5
0
1
2
3
4
5
6
7
8
9
10
VM pengamat keadaan normal
VM pengamat 2 VM collision
VM pengamat ke OpenWrt

Lampiran 5. Lingkungan fisik – Keadaan Normal

Fisik (normal) ke router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	1	26	1
2	NodeMCU	Router	1	7	1
3	NodeMCU	Router	1	45	1
4	NodeMCU	Router	1	17	1
5	NodeMCU	Router	1	23	1
6	NodeMCU	Router	1	23	1
7	NodeMCU	Router	1	9	1
8	NodeMCU	Router	1	1	1
9	NodeMCU	Router	1	12	1
10	NodeMCU	Router	1	56	1
Keberhasilan (%)			100		100
Rata-rata				22.1	
Standar deviasi				16.0714	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	1	31.2	1
2	Android	Router	1	77.5	1
3	Android	Router	1	31.5	1
4	Android	Router	1	25.7	1
5	Android	Router	1	37.2	1
6	Android	Router	1	17.3	1
7	Android	Router	1	52.9	1
8	Android	Router	1	22.2	1
9	Android	Router	1	21.8	1
10	Android	Router	1	24.2	1
Keberhasilan (%)			100		100
Rata-rata				34.15	
Standar deviasi				17.33628	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	1	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	1	1
4	Komputer	Router	1	1	1
5	Komputer	Router	1	1	1
6	Komputer	Router	1	1	1
7	Komputer	Router	1	1	1
8	Komputer	Router	1	1	1
9	Komputer	Router	1	1	1
10	Komputer	Router	1	4	1
Keberhasilan (%)			100		100
Rata-rata				1.3	
Standar deviasi				0.9	

Fisik (normal) ke komputer

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	1	20	1
2	NodeMCU	Komputer	1	26	1
3	NodeMCU	Komputer	1	13	1
4	NodeMCU	Komputer	1	27	1
5	NodeMCU	Komputer	1	17	1
6	NodeMCU	Komputer	1	20	1
7	NodeMCU	Komputer	1	19	1
8	NodeMCU	Komputer	1	42	1
9	NodeMCU	Komputer	1	47	1
10	NodeMCU	Komputer	1	11	1
Keberhasilan (%)			100		100
Rata-rata				24.2	
Standar deviasi				11.2321	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Komputer	1	304	1
2	Android	Komputer	1	25.5	1
3	Android	Komputer	1	77.8	1
4	Android	Komputer	1	281.5	1
5	Android	Komputer	1	35.2	1
6	Android	Komputer	1	136.8	1
7	Android	Komputer	1	97.2	1
8	Android	Komputer	1	234.4	1
9	Android	Komputer	1	243.2	1
10	Android	Komputer	1	48.9	1
Keberhasilan (%)			100		100
Rata-rata				148.43	
Standar deviasi				101.9301	

Fisik (normal) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	1	248	1
2	NodeMCU	Android	1	364	1
3	NodeMCU	Android	1	202	1
4	NodeMCU	Android	1	278	1
5	NodeMCU	Android	1	396	1
6	NodeMCU	Android	1	182	1
7	NodeMCU	Android	1	269	1
8	NodeMCU	Android	1	265	1
9	NodeMCU	Android	1	358	1
10	NodeMCU	Android	1	92	1
Keberhasilan (%)			100		100
Rata-rata				265.4	
Standar deviasi				87.67577	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	1	316	1
2	Komputer	Android	1	301	1
3	Komputer	Android	1	73	1
4	Komputer	Android	1	447	1
5	Komputer	Android	1	351	1
6	Komputer	Android	1	66	1
7	Komputer	Android	1	361	1
8	Komputer	Android	1	46	1
9	Komputer	Android	1	204	1
10	Komputer	Android	1	171	1
Keberhasilan (%)			100		100
Rata-rata				233.6	
Standar deviasi				134.4457	

Standar deviasi pada keadaan normal sudah cukup tinggi

Lampiran 6. Lingkungan fisik - NodeMCU

Fisik (NodeMCU router) ke router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	0		0
2	NodeMCU	Router	0		0
3	NodeMCU	Router	0		0
4	NodeMCU	Router	0		0
5	NodeMCU	Router	0		0
6	NodeMCU	Router	0		0
7	NodeMCU	Router	0		0
8	NodeMCU	Router	0		0
9	NodeMCU	Router	0		0
10	NodeMCU	Router	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	1	24	1
2	Android	Router	1	38.9	1
3	Android	Router	1	23.2	1
4	Android	Router	1	55	1
5	Android	Router	1	2.02	1
6	Android	Router	1	34.9	1
7	Android	Router	1	60.3	1
8	Android	Router	1	24.5	1
9	Android	Router	1	55.4	1
10	Android	Router	1	21.7	1
Keberhasilan (%)			100		100
Rata-rata				33.992	
Standar deviasi				17.5965	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	70	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	2	1
4	Komputer	Router	1	2	1
5	Komputer	Router	1	2	1
6	Komputer	Router	1	1	1
7	Komputer	Router	1	2	1
8	Komputer	Router	1	97	1
9	Komputer	Router	1	2	1
10	Komputer	Router	1	2	1
Keberhasilan (%)			100		100
Rata-rata				18.1	
Standar deviasi				33.25492	

Fisik (NodeMCU router) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	0		0
2	NodeMCU	Android	0		0
3	NodeMCU	Android	0		0
4	NodeMCU	Android	0		0
5	NodeMCU	Android	0		0
6	NodeMCU	Android	0		0
7	NodeMCU	Android	0		0
8	NodeMCU	Android	0		0
9	NodeMCU	Android	0		0
10	NodeMCU	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	0		0
2	Komputer	Android	0		0
3	Komputer	Android	0		0
4	Komputer	Android	0		0
5	Komputer	Android	0		0
6	Komputer	Android	0		0
7	Komputer	Android	0		0
8	Komputer	Android	0		0
9	Komputer	Android	0		0
10	Komputer	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Fisik (NodeMCU router) ke Komputer

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	0		0
2	NodeMCU	Komputer	0		0
3	NodeMCU	Komputer	0		0
4	NodeMCU	Komputer	0		0
5	NodeMCU	Komputer	0		0
6	NodeMCU	Komputer	0		0
7	NodeMCU	Komputer	0		0
8	NodeMCU	Komputer	0		0
9	NodeMCU	Komputer	0		0
10	NodeMCU	Komputer	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Komputer	1	61.6	1
2	Android	Komputer	1	23.1	1
3	Android	Komputer	1	51.5	1
4	Android	Komputer	1	31.9	1
5	Android	Komputer	1	91.4	1
6	Android	Komputer	1	75.3	1
7	Android	Komputer	1	62.5	1
8	Android	Komputer	1	87.1	1
9	Android	Komputer	1	19.6	1
10	Android	Komputer	1	76	1
Keberhasilan (%)			100		100
Rata-rata				58	
Standar deviasi				24.56603	

Fisik (NodeMCU router) ke Router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	1	25	0
2	NodeMCU	Router	1	63	0
3	NodeMCU	Router	1	19	0
4	NodeMCU	Router	1	7	0
5	NodeMCU	Router	1	38	0
6	NodeMCU	Router	1	22	0
7	NodeMCU	Router	1	58	0
8	NodeMCU	Router	1	15	0
9	NodeMCU	Router	1	56	0
10	NodeMCU	Router	1	13	0
Keberhasilan (%)			100		0
Rata-rata				31.6	
Standar deviasi				19.59694	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	0		1
2	Android	Router	0		1
3	Android	Router	0		1
4	Android	Router	0		1
5	Android	Router	0		1
6	Android	Router	0		1
7	Android	Router	0		1
8	Android	Router	0		1
9	Android	Router	0		1
10	Android	Router	0		1
Keberhasilan (%)			0		100
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	33	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	1	1
4	Komputer	Router	1	1	1
5	Komputer	Router	1	1	1
6	Komputer	Router	1	1	1
7	Komputer	Router	1	84	1
8	Komputer	Router	1	2	1
9	Komputer	Router	1	1	1
10	Komputer	Router	1	1	1
Keberhasilan (%)			100		100
Rata-rata				12.6	
Standar deviasi				25.62889	

Fisik (NodeMCU router) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	0		0
2	NodeMCU	Android	0		0
3	NodeMCU	Android	0		0
4	NodeMCU	Android	0		0
5	NodeMCU	Android	0		0
6	NodeMCU	Android	0		0
7	NodeMCU	Android	0		0
8	NodeMCU	Android	0		0
9	NodeMCU	Android	0		0
10	NodeMCU	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	0		1
2	Komputer	Android	0		1
3	Komputer	Android	0		1
4	Komputer	Android	0		1
5	Komputer	Android	0		1
6	Komputer	Android	0		1
7	Komputer	Android	0		1
8	Komputer	Android	0		1
9	Komputer	Android	0		1
10	Komputer	Android	0		1
Keberhasilan (%)			0		100
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Fisik (NodeMCU router) ke Komputer

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	1	82	0
2	NodeMCU	Komputer	1	77	0
3	NodeMCU	Komputer	1	65	0
4	NodeMCU	Komputer	1	61	0
5	NodeMCU	Komputer	1	64	0
6	NodeMCU	Komputer	1	44	0
7	NodeMCU	Komputer	1	42	0
8	NodeMCU	Komputer	1	33	0
9	NodeMCU	Komputer	1	22	0
10	NodeMCU	Komputer	1	45	0
Keberhasilan (%)			100		0
Rata-rata				53.5	
Standar deviasi				18.35892	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	0		1
2	Komputer	Android	0		1
3	Komputer	Android	0		1
4	Komputer	Android	0		1
5	Komputer	Android	0		1
6	Komputer	Android	0		1
7	Komputer	Android	0		1
8	Komputer	Android	0		1
9	Komputer	Android	0		1
10	Komputer	Android	0		1
Keberhasilan (%)			0		100
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Fisik (NodeMCU router) ke Router

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Router	0		1
2	NodeMCU	Router	0		1
3	NodeMCU	Router	0		1
4	NodeMCU	Router	0		1
5	NodeMCU	Router	0		1
6	NodeMCU	Router	0		1
7	NodeMCU	Router	0		1
8	NodeMCU	Router	0		1
9	NodeMCU	Router	0		1
10	NodeMCU	Router	0		1
Keberhasilan (%)			0		100
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Router	1	28.3	1
2	Android	Router	1	14.7	1
3	Android	Router	1	369	1
4	Android	Router	1	1.62	1
5	Android	Router	1	4.36	1
6	Android	Router	1	61.87	1
7	Android	Router	1	4.72	1
8	Android	Router	1	230	1
9	Android	Router	1	28	1
10	Android	Router	1	59.03	1
Keberhasilan (%)			100		100
Rata-rata				80.16	
Standar deviasi				115.742	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Router	1	1	1
2	Komputer	Router	1	1	1
3	Komputer	Router	1	1	1
4	Komputer	Router	1	1	1
5	Komputer	Router	1	1	1
6	Komputer	Router	1	1	1
7	Komputer	Router	1	1	1
8	Komputer	Router	1	1	1
9	Komputer	Router	1	1	1
10	Komputer	Router	1	1	1
Keberhasilan (%)			100		100
Rata-rata				1.2	
Standar deviasi				0.6	

Fisik (NodeMCU router) ke Android

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Android	0		0
2	NodeMCU	Android	0		0
3	NodeMCU	Android	0		0
4	NodeMCU	Android	0		0
5	NodeMCU	Android	0		0
6	NodeMCU	Android	0		0
7	NodeMCU	Android	0		0
8	NodeMCU	Android	0		0
9	NodeMCU	Android	0		0
10	NodeMCU	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

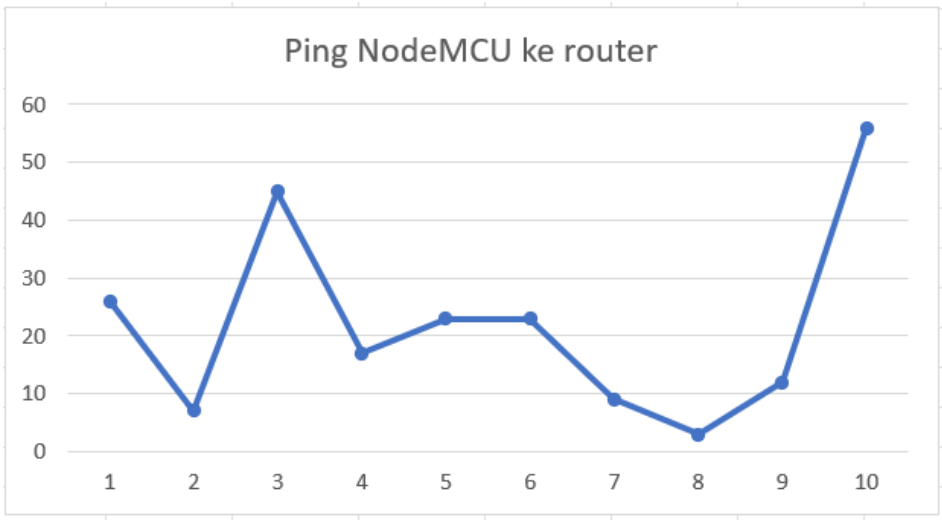
Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Komputer	Android	0		0
2	Komputer	Android	0		0
3	Komputer	Android	0		0
4	Komputer	Android	0		0
5	Komputer	Android	0		0
6	Komputer	Android	0		0
7	Komputer	Android	0		0
8	Komputer	Android	0		0
9	Komputer	Android	0		0
10	Komputer	Android	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

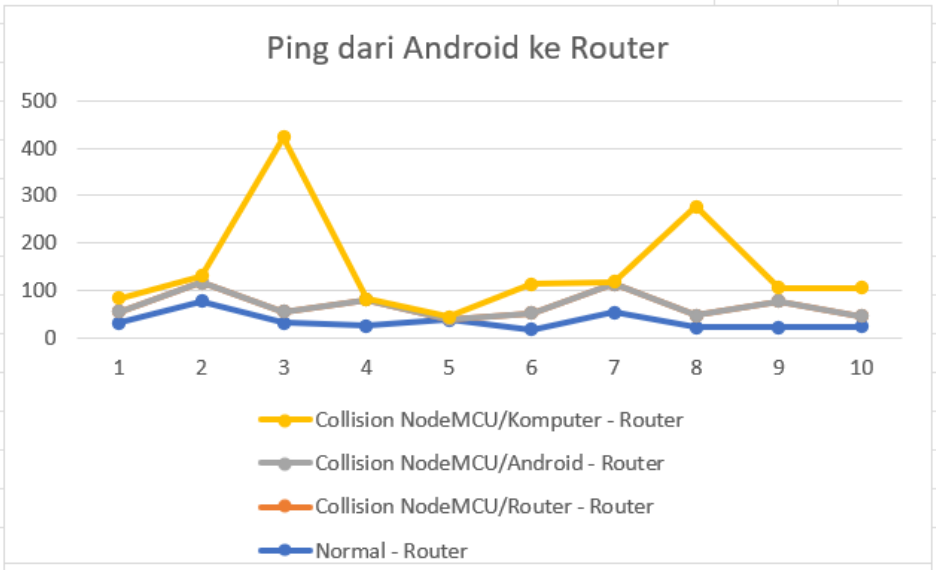
Fisik (NodeMCU router) ke Komputer

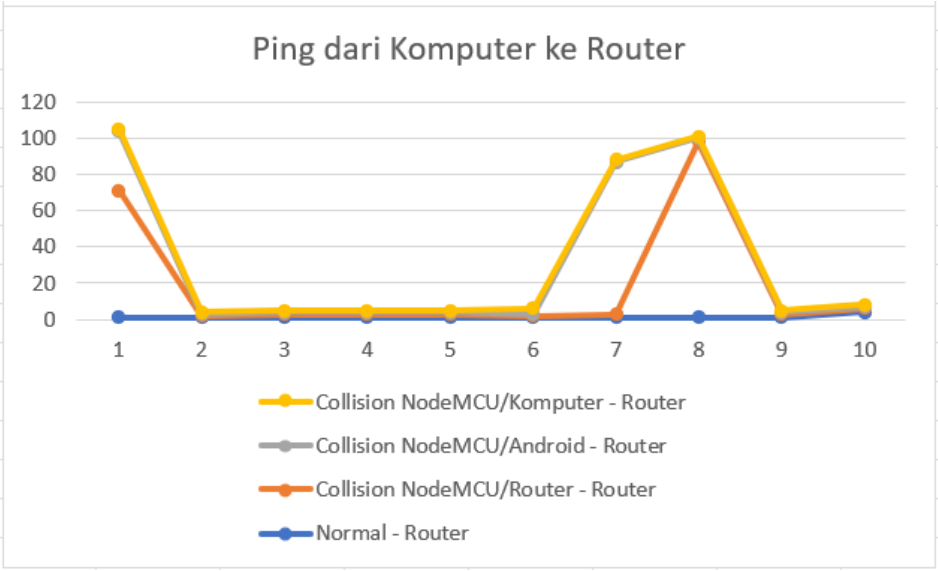
Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	NodeMCU	Komputer	0		0
2	NodeMCU	Komputer	0		0
3	NodeMCU	Komputer	0		0
4	NodeMCU	Komputer	0		0
5	NodeMCU	Komputer	0		0
6	NodeMCU	Komputer	0		0
7	NodeMCU	Komputer	0		0
8	NodeMCU	Komputer	0		0
9	NodeMCU	Komputer	0		0
10	NodeMCU	Komputer	0		0
Keberhasilan (%)			0		0
Rata-rata				#DIV/0!	
Standar deviasi				#DIV/0!	

Pengujian ke	Dari	Tujuan	PING		HTTP GET
			Berhasil	Latency	Berhasil
1	Android	Komputer	1	142	0
2	Android	Komputer	1	53.5	0
3	Android	Komputer	1	124	0
4	Android	Komputer	1	291	0
5	Android	Komputer	1	115.02	0
6	Android	Komputer	1	186.54	0
7	Android	Komputer	1	149.57	0
8	Android	Komputer	1	75.91	0
9	Android	Komputer	1	245.75	0
10	Android	Komputer	1	188.63	0
Keberhasilan (%)			100		0
Rata-rata				157.192	
Standar deviasi				69.37191	

HTTP GET pada Android - Komputer menampilkan respon HTTP dari NodeMCU


Ping NodeMCU ke router
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10


Ping dari Android ke Router
500
400
300
200
100
0
1
2
3
4
5
6
7
8
9
10
Collision NodeMCU/Komputer - Router
Collision NodeMCU/Android - Router
Collision NodeMCU/Router - Router
Normal - Router


Ping dari Komputer ke Router
120
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9
10
Collision NodeMCU/Komputer - Router
Collision NodeMCU/Android - Router
Collision NodeMCU/Router - Router
Normal - Router

Lampiran 7. Kode Program NodeMCU

```
#include <ESP8266WiFi.h>
#include <ESP8266Ping.h>
#include <ESP8266HTTPClient.h>
#include <WiFiUDP.h>
#include <ESP8266WebServer.h>
const char * ssid = ""; //ganti nama hotspot
const char * pass = ""; //ganti password
//IPAddress localIP(192, 168, 1, 11);       // The desired IP address for the NodeMCU
//IPAddress gateway(192, 168, 1, 1);        // IP address of the gateway/router
      //IPAddress subnet(255, 255, 255, 0);       // Subnet mask
      //IPAddress dns(8, 8, 8, 8);              // DNS server address
      WiFiClient client;
      ESP8266WebServer server(80);
      // 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
      //04:E5:98:32:D0:1E
      uint8_t newMACAddress[] = {
        0x34,
        0x13,
        0xE8,
        0xC6,
        0x4C,
        0x42
      }; // MAC router (uji collision 1)
      // ping ec2.me-south-1.amazonaws.com 99.82.132.91
      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");
```

```
39.         } else {
40.           Serial.println("Ping failed!");
41.         }
42.       }
43.       void do_curl(const char * url) {
44.         HTTPClient http;
45.         Serial.print("HTTP GET request to: ");
46.         Serial.println(url);
47.         http.begin(client, url); // Specify the URL
48.         int httpCode = http.GET(); // Perform the GET request
49.         if (httpCode > 0) { // Check for a successful request
50.           Serial.printf("HTTP   GET   request   successful.
   Response code: %d\n", httpCode);
51.           String  payload  =  http.getString();  //  Get  the
   response payload (HTML content)
52.           Serial.println("Response (first 10 characters):");
53.           Serial.println(payload.substring(0, 10));
54.         } else {
55.           Serial.printf("HTTP  GET  request  failed.  Error
   code: %d\n", httpCode);
56.         }
57.         http.end(); // Close the connection
58.       }
59.       void  traceroute(uint8_t  a,  uint8_t  b,  uint8_t  c,
   uint8_t d) {
60.         IPAddress destinationIP(a, b, c, d); // Target
61.         Serial.println("Traceroute:");
62.         int udpPort = 33434;
63.         const int maxHops = 30; // Maximum number of hops to
   attempt
64.         for (int ttl = 1; ttl <= maxHops; ttl++) {
65.           WiFiUDP udp;
66.           IPAddress hopIP = udp.remoteIP();
67.           udp.beginPacketMulticast(destinationIP,   udpPort,
   WiFi.localIP(), ttl);
68.           udp.write("Traceroute");
69.           udp.endPacket();
70.           Serial.print("Hop ");
71.           Serial.print(ttl);
72.           Serial.print(": ");
73.           int packetSize = udp.parsePacket();
74.           if (packetSize) {
75.             Serial.print(hopIP);
76.           } else {
77.             Serial.print("*");
78.           }
79.           Serial.println();
80.           // Stop tracing if the destination is reached
81.           if (hopIP == destinationIP) {
```

```
82.             break;
83.           }
84.         }
85.       }
86.       // run HTTP server
87.       void http_server() {
88.         server.send(200, "text/plain", "Hello World!");
89.       }
90.       void setup() {
91.         Serial.begin(115200);
92.         delay(10);
93.         Serial.print(" Connect to : ");
94.         Serial.println(ssid);
95.         WiFi.begin(ssid, pass);
96.         while (WiFi.status() != WL_CONNECTED) {
97.           delay(500);
98.           Serial.print("….");
99.         }
100.        Serial.print("\n");
101.        Serial.print("IP address : ");
102.        Serial.print(WiFi.localIP());
103.        Serial.print("\n");
104.        Serial.print("Connect to : ");
105.        Serial.println(ssid);
106.        Serial.print("\n");
107.        Serial.println(WiFi.macAddress());
108.        wifi_set_macaddr(STATION_IF, & newMACAddress[0]);
109.        Serial.print("\n");
110.        Serial.println(WiFi.macAddress());
111.        server.on("/", http_server);
112.        server.begin();
113.        Serial.println("HTTP server started.");
114.      }
115.      void loop() {
116.          // Uji HTTP server
117.          server.handleClient();
118.          /*
119.          // Uji ping dan HTTP GET
120.          do_ping(192,168,1,11);
121.          do_ping(192,168,1,1);
122.          do_ping(192,168,1,74);
123.          do_ping(192,168,1,99);
124.          do_curl("http://192.168.1.11"); // Alamat NodeMCU
125.          do_curl("http://192.168.1.1"); // Alamat router
126.          do_curl("http://192.168.1.74:1111");   //   Alamat
    Android
127.          do_curl("http://192.168.1.99"); // Alamat komputer
128.          // traceroute(192,168,1,1);
129.          // traceroute(192,168,1,99);
```

```
130.          delay(1000);
131.          */
132.      }
```