NETWORK INTRUSION DETECTION SYSTEM UNTUK SISTEM IOT BERBASIS HETEROGENEOUS COMPUTING

TUGAS AKHIR

Disusun dalam rangka memenuhi salah satu persyaratan

Untuk menyelesaikan program Strata-1 Departemen Teknik Informatika

Fakultas Teknik Universitas Hasanuddin

Makassar

Disusun Oleh:

MUHAMMAD ILHAM ASKARI

D121171320

DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

LEMBAR PENGESAHAN SKRIPSI

NETWORK INTRUSION DETECTION SYSTEM UNTUK SISTEM IOT BERBASIS HETEROGENEOUS COMPUTING

Disusun dan diajukan oleh

MUHAMMAD ILHAM ASKARI

D121171320

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika Fakultas Teknik Universitas Hasanuddin pada tanggal 19 Oktober 2022 dan dinyatakan telah memenuhi syarat kelulusan.

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
Dr.Eng. Ady Wahyudi Paundu, ST.,MT.	Muhammad Alief Fadhal Imran Oemar, S.T., M.Sc
Nip. 197503132009121003	Nip. 199405222022043001

Ketua Program Studi,

Dr. Indrabayu, S.T., M.T., M.Bus.Sys.

Nip. 19750716 200212 1 004

PERNYATAAN KEASLIAN

Yang bertanda tangan di bawah ini:

NAMA	: Muhammad Ilham Askari
NIM	: D121171320
Departemen	: Teknik Informatika
Jenjang	: S1

Menyatakan dengan ini karya tulisan saya berjudul:

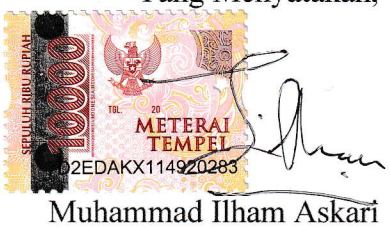
NETWORK INTRUSION DETECTION SYSTEM UNTUK SISTEM IOT BERBASIS HETEROGENEOUS COMPUTING

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilalihan tulisan orang lain, bahwa skripsi yang saya tulis benar-benar merupakan hasil karya saya sendiri.

Apabila di kemudian hari terbukti atau dapat dibuktikan bahwa Sebagian atau keseluruhan skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 5 Oktober 2022

Yang Menyatakan,



Muhammad Ilham Askari

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan yang Maha Esa atas berkat, rahmat, karunia, dan hidayah-Nya sehingga tugas akhir yang berjudul "NETWORK INTRUSION DETECTION SYSTEM UNTUK SISTEM IOT BERBASIS HETEROGENEOUS COMPUTING" dapat diselesaikan dengan baik sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Penulis menyadari bahwa dalam menyusun dan menulis tugas akhir ini tidak lepas dari bantuan, bimbingan, serta dukungan dari berbagai pihak, dari masa perkuliahan hingga penyusunan tugas akhir. Oleh sebab itu, penulis menyampaikan terima kasih kepada:

1. Kedua orang tua penulis, Bapak H. Kamaruddin dan Ibu Hj. Andi Astuti yang selalu memberikan yang doa terbaik, semangat terbaik, serta dukungan terbaik kepada penulis.
2. Bapak Dr. Eng. Ady Wahyudi Paundu, S.T., M.T., selaku pembimbing utama dan Bapak Muhammad Alief Fahdal Imran Oemar, S.T., M.Sc., selaku pembimbing pendamping yang senantiasa menyediakan waktu, tenaga, pikiran, dan perhatiannya dalam mengarahkan penulis untuk menyelesaikan tugas akhir.
3. Bapak Dr. Indrabayu, S.T., M.T., M.Bus.Sys. selaku Ketua Departemen Teknik Informatika atas segala bimbingan dan dukungan selama masa perkuliahan.

4. Ibu Anugrayani Bustamin, S.T., M.T., selaku Dosen Pembimbing Akademik penulis yang senantiasa membimbing dan menyediakan waktu, tenaga, dan pikirannya selama masa perkuliahan penulis.
5. Segenap Dosen dan Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah sangat banyak membantu selama masa perkuliahan dan penyelesaian tugas akhir penulis.
6. Seluruh warga Organisasi Kemahasiswaan Informatika Fakultas Teknik Universitas Hasanuddin atas pembelajaran dan pengalaman yang penulis dapatkan selama masa perkuliahan.
7. Teman-teman RECOGN17ER atas segala kenangan, semangat, dan bantuannya selama ini.
8. Teman-teman PROTON yang masih dan akan selalu saling membantu, menyemangati, dan menemani.
9. Seluruh pihak yang tidak bisa saya sebutkan namanya satu per satu, tanpa pernah bertemu dan mengenal kalian, tidak akan ada pengalaman yang dapat menjadi guru saya.

Akhir kata, penulis menyadari bahwa tugas akhir ini masih jauh dari kata sempurna, oleh karenanya diharapkan segala bentuk saran serta masukan dari berbagai pihak. Semoga tugas akhir ini dapat memberikan manfaat dan kontribusi yang besar bagi kepentingan bersama.

Gowa, 5 Oktober 2022

Penulis

ABSTRAK

Teknologi *Internet of Things* berkembang dengan sangat cepat, dan kini sudah digunakan di berbagai jenis bidang seperti medis, pemantauan lingkungan, otomasi rumah (rumah cerdas), transportasi cerdas, dan berbagai utilitas lainnya dalam industri. Hal ini menyebabkan meningkatnya jumlah perangkat IoT baik itu dalam industri, maupun pada kehidupan sehari-hari. Hal ini meningkatkan risiko terhadap serangan siber. Sistem yang mampu membantu melindungi dari serangan siber terhadap perangkat yang terhubung ke internet disebut dengan *Network Intrusion Detection System*. Sistem ini bekerja dengan memindai arus lalu lintas jaringan yang melalui sebuah perangkat untuk mendeteksi adanya indikasi serangan atau pelanggaran. Untuk memberikan perlindungan terhadap perangkat IoT dari serangan siber, maka dikembangkanlah sebuah purwarupa sederhana dari NIDS yang mampu menggunakan GPU dalam mendeteksi pola serangan yang ada. NIDS yang digunakan sebagai purwarupa adalah Snort yang menggunakan algoritma Aho-Corasick dalam melakukan deteksi pola. Snort kemudian dimodifikasi untuk melakukan komputasi pada GPU dengan menggunakan kerangka kerja OpenCL. Proses deteksi pola akan dilakukan oleh GPU dengan memanfaatkan konsep *General Purpose Computing on Graphics Processing Unit* atau GPGPU. Sistem ini kemudian diimplementasikan pada *Single Board Computer* yang umum digunakan sebagai IoT *Gateway* pada sebuah sistem IoT yaitu Odroid N2+. Hasil uji coba purwarupa menunjukkan peningkatan performa pada kemampuan NIDS dalam mendeteksi pola serangan.

Kata kunci: *GPGPU, Aho-Corasick, Internet of Things, Network Intrusion Detection System*

ABSTRACT

The Internet of Things is rapidly spreading, reaching a lot of domains like medical devices, environmental monitoring, home automation, smart transportation, etc. This increases the amount of IoT devices either in industrial environments or our daily lives. This also increases the risk of cyber-attacks. Systems that can protect these kinds of devices are called Network Intrusion Detection Systems. The system works by scanning the network traffic that comes from the connected devices to detect if there are any signs of attacks or intrusion. To protect IoT devices from cyber-attacks, there is a need to develop a simple prototype of NIDS that can utilize a GPU to detect patterns of attack. NIDS that will be used to create the prototype is Snort that uses Aho-Corasick algorithm to do pattern detection. It is then modified to compute on the GPU using the OpenCL framework. Pattern detection will be performed on the GPU using the concept of General-Purpose Computing on Graphics Processing Unit or GPGPU. This system will be implemented on a Single Board Computer that is used commonly as an IoT Gateway on an IoT System. The result shows that the prototype can increase the performance of NIDS on the ability to detect a pattern of attack.

Keywords: *GPGPU, Aho-Corasick, Internet of Things, Network Intrusion Detection System*

DAFTAR ISI

DAFTAR ISI i

DAFTAR GAMBAR iii

DAFTAR TABEL iv

BAB 1 PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Rumusan Masalah 2

1.3 Tujuan Penelitian 3

1.4 Manfaat Penelitian 3

1.5 Batasan Masalah 3

1.6 Sistematika Penulisan 4

BAB 2 TINJAUAN PUSTAKA 5

2.1 Intrusion Detection System (IDS) 5

2.2 Snort 7

2.2.1 Desain dan Arsitektur Snort 8

2.3 Graphics Processing Unit 12

2.3.1 General-purpose Computing on Graphics Processing Units 13

2.4 Algoritma Aho-Corasick 14

2.4.1 Parallel Failureless Aho-Corasick 15

2.5 OpenCL 16

2.6 Dataset IoT-23 19
2.7 Basic Instrumentation Profiler 21
BAB 3 METODOLOGI PENELITIAN 22
3.1 Tahapan Penelitian 22
3.2 Waktu dan Lokasi Penelitian 24
3.3 Instrumen Penelitian 24
3.4 Perancangan Sistem 25
3.5 Implementasi Sistem 27
3.5.1 Konversi State 28
3.5.2 Pencocokan Pola pada GPU 30
3.6 Pengujian Sistem 31
BAB 4 HASIL DAN PEMBAHASAN 34
4.1 Hasil Uji Dataset bigFlows 34
4.1.1 Hasil Profiling Snort GPU dengan Dataset bigFlows 37
4.2 Hasil Uji Dataset IoT-23 41
BAB 5 KESIMPULAN DAN SARAN 43
5.1 Kesimpulan 43
5.2 Saran 44
DAFTAR PUSTAKA 45
LAMPIRAN 48

DAFTAR GAMBAR

Gambar 2.1 NIDS vs HIDS..... 6

Gambar 2.2 Proses Analisis Snort..... 9

Gambar 2.3 Alur Kerja GPGPU 13

Gambar 2.4 Contoh State Machine 15

Gambar 2.5 Pola akses PFAC 16

Gambar 2.6 Arsitektur OpenCL..... 17

Gambar 3.1 Tahapan Penelitian 22

Gambar 3.2 Alur Kerja Snort..... 26

Gambar 3.3 Alur Kerja Sistem..... 27

Gambar 3.4 Alur Konversi State..... 29

Gambar 3.5 Hasil Konversi State menjadi Himpunan 1-Dimensi 30

Gambar 3.6 Kernel Aho-Corasick pada GPU 31

Gambar 3.7 Luaran clinfo 31

Gambar 4.1 Waktu Eksekusi Snort dengan Heterogeneous Computing 36

Gambar 4.2 Perbandingan Waktu Eksekusi Snort 37

Gambar 4.3 Hasil Profiling AC_GPU pada modifikasi Snort 37

Gambar 4.4 Perbandingan Waktu Transfer Data ke GPU 38

Gambar 4.5 Perbandingan Waktu Transfer Data dari GPU..... 39

Gambar 4.6 Perbandingan Waktu Deteksi Pola pada GPU 40

Gambar 4.7 Perbandingan Waktu Eksekusi Fungsi AC_GPU 40

DAFTAR TABEL

Tabel 2.1 Contoh Tabel Transisi Full Matrix 11

Tabel 2.2 Contoh Tabel Transisi Sparse Matrix 11

Tabel 2.3 Contoh Tabel Transisi Banded Row 11

Tabel 2.4 Contoh Tabel Transisi Sparse Bands 11

Tabel 2.5 Data Lalu Lintas Jaringan Perangkat IoT yang Terinfeksi 20

Tabel 2.6 Data Lalu Lintas Jaringan Perangkat IoT Normal 20

Tabel 3.1 Parameter Pengujian 32

Tabel 3.2 Dataset Pengujian..... 32

Tabel 4.1 Hasil Uji Dataset bigFlows 34

Tabel 4.2 Hasil Uji Dataset IoT-23 41

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Perangkat IoT kini semakin marak digunakan di berbagai jenis bidang termasuk perangkat medis, pemantauan lingkungan, otomasi rumah (rumah cerdas), transportasi cerdas, dan berbagai macam penerapan unik lainnya. Hal ini menyebabkan meningkatnya perangkat IoT yang kita temui dalam kehidupan sehari-hari (Meneghello et al., 2019). Diprediksi bahwa jumlah perangkat yang terhubung ke internet pada tahun 2023 akan mencapai 29,3 miliar perangkat, meningkat 59% dibandingkan dengan jumlah perangkat yang terhubung ke internet pada tahun 2018 yaitu 18,4 miliar perangkat. Jumlah perangkat IoT pada tahun 2018 mencapai sekitar 6 miliar perangkat atau 33% dari keseluruhan perangkat yang terhubung ke internet pada tahun 2018 dan diprediksi akan meningkat hingga 14,7 miliar perangkat atau 50% dari keseluruhan perangkat yang terhubung ke internet pada tahun 2023 (Cisco Public, 2020).

Namun, peningkatan penerapan perangkat cerdas juga membawa beberapa risiko khususnya dalam keamanan dan privasi. Risiko ini muncul akibat dari ranah penerapan perangkat cerdas ini cenderung berkaitan dengan hal-hal yang penting dalam kehidupan sehari-hari seperti kunci cerdas, televisi cerdas, alarm cerdas, dan lain-lain. Terlepas dari ranah penerapannya yang sangat membantu, banyaknya perangkat IoT itu sendiri merupakan hal yang sangat memikat bagi para aktor siber untuk dapat diakuisisi dan digunakan untuk melakukan serangan DDoS (*Distributed Denial of Service*) terhadap suatu target (Kolias et al., 2017).

Untuk mengurangi risiko tersebut, *Network Intrusion Detection System* (NIDS) dapat diterapkan dalam suatu jaringan IoT untuk memberikan perlindungan tingkat jaringan untuk perangkat pintar yang digunakan (Neshenko et al., 2019). Sistem ini memantau aktivitas jaringan dari perangkat yang terhubung ke internet dan juga akan memberikan peringatan apabila terdapat aktivitas yang mencurigakan dan/atau berbahaya. Namun, penerapan sistem ini dalam konteks IoT merupakan sebuah tantangan tersendiri. Performa dari suatu NIDS sangat dipengaruhi oleh kemampuan komputasi CPU dari perangkat pengaplikasiannya. Dalam konteks perangkat IoT, umumnya mengutamakan efisiensi dari penggunaan daya dibandingkan kekuatan komputasinya sehingga dapat menghambat performa dari NIDS itu sendiri. Sedangkan dalam suatu sistem IoT, perangkat yang umum digunakan sebagai *Gateway* yaitu Home Assistant Blue yang menggunakan Odroid N2+ sebagai perangkat kerasnya dan menggunakan Linux sebagai perangkat lunak memiliki sumber daya GPU yang tidak termanfaatkan. Oleh karena itu, penulis mengusulkan tugas akhir yang berjudul "Network Intrusion Detection System untuk Sistem IoT berbasis Heterogeneous Computing" yang akan menghasilkan sebuah purwarupa sederhana dari NIDS yang dimodifikasi agar dapat menggunakan sumber daya CPU dan GPU secara bersamaan (Heterogeneous) untuk mengoptimalkan kinerja NIDS.

1.2 Rumusan Masalah

1. Bagaimana implementasi NIDS berbasis *Heterogeneous Computing* yang optimal untuk diterapkan pada sistem IoT?
2. Bagaimana performansi NIDS apabila diterapkan pada perangkat IoT?

3. Bagaimana perbandingan performansi NIDS dalam mode komputasi tradisional (*Homogeneous*) dengan mode komputasi *Heterogeneous* pada sistem IoT?

1.3 Tujuan Penelitian

Tujuan dari pengerjaan tugas akhir ini adalah:

1. Mengimplementasikan NIDS berbasis *Heterogeneous Computing* dengan membangun sebuah purwarupa sederhana.
2. Menguji performansi NIDS pada sistem IoT.
3. Membandingkan performansi NIDS dalam mode komputasi tradisional (*Homogeneous*) dengan mode komputasi *Heterogeneous* pada sistem IoT.

1.4 Manfaat Penelitian

Manfaat dari tugas akhir ini adalah:

1. Memberikan informasi terhadap bidang ilmu pengetahuan yang relevan sesuai dengan tema dari tugas akhir ini.
2. Membantu pengembangan sistem NIDS dengan mode komputasi heterogen untuk memanfaatkan sumber daya GPU yang ada pada Odroid N2+ dan berbagai *Single Board Computer* (SBC) lainnya.

1.5 Batasan Masalah

1. *Single Board Computer* yang digunakan memiliki sumber daya GPU dan mendukung kerangka kerja OpenCL.

2. Uji coba dilakukan dengan menyimpan data pada NFS (*Network File System*) dikarenakan keterbatasan penyimpanan dengan ukuran dataset yang besar (62,9 GB).

1.6 Sistematika Penulisan

Berikut adalah gambaran singkat mengenai isi tulisan secara keseluruhan:

BAB I PENDAHULUAN

Bab ini berisi latar belakang masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan landasan teori yang menunjang setiap elemen percobaan yang dilakukan.

BAB III METODOLOGI PENELITIAN

Bab ini berisi tahapan penelitian, instrumen yang digunakan, perancangan sistem, implementasi sistem, dan skenario pengujian kinerja sistem terhadap dataset *bigFlows* dan IoT-23.

BAB IV HASIL DAN PEMBAHASAN

Bab ini menjelaskan secara keseluruhan hasil pengujian *Network Intrusion Detection System* untuk sistem IoT berbasis *Heterogeneous Computing*.

BAB V PENUTUP

Bab ini berisi kesimpulan dari hasil penelitian dan uraian saran untuk pengembangan lebih lanjut.

BAB 2

TINJAUAN PUSTAKA

2.1 Intrusion Detection System (IDS)

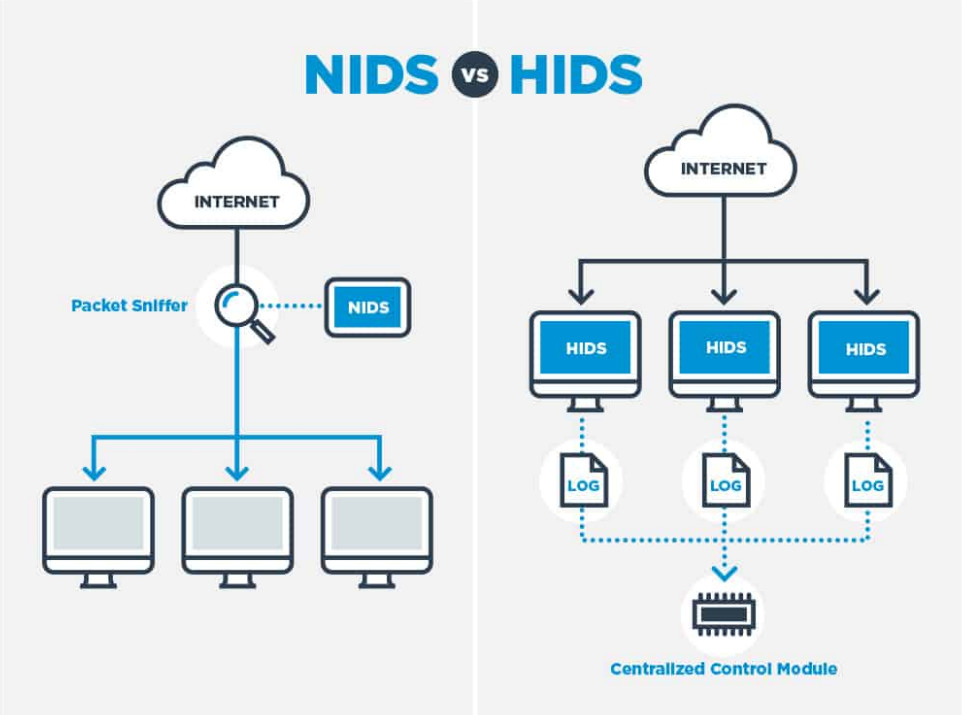
IDS adalah sebuah sistem yang bekerja untuk memonitor kejadian pada sebuah jaringan komputer dan sekaligus menganalisis masalah keamanan jaringan. IDS bekerja dengan memonitor suatu sistem dapat mendeteksi perilaku yang tidak lazim (anomali), kegiatan yang mencurigakan, mendeteksi serangan dan menyajikan informasi mengenai sistem yang dipantau (Prayogo, 2021). IDS jika ditinjau dari kemampuannya dalam mendeteksi serangan di dalam jaringan dikelompokkan menjadi dua yaitu:

a. *Network Intrusion Detection System* (NIDS)

   NIDS ini bekerja dengan menganalisis lalu lintas jaringan untuk mencari adanya usaha serangan atau penyusupan ke dalam sistem jaringan. Jadi NIDS ini dapat melindungi sebuah sistem jaringan.

b. *Host Intrusion Detection System* (HIDS)

   HIDS ini bekerja dengan menganalisis lalu lintas jaringan yang masuk dan keluar pada satu perangkat, untuk mencari adanya usaha serangan atau penyusupan ke dalam perangkat itu. Jika dibandingkan dengan NIDS, HIDS hanya dapat melindungi satu perangkat saja.



Gambar 2.1 NIDS vs HIDS

Sedangkan IDS jika ditinjau dari metode yang digunakan dalam mendeteksi pola dari serangan dapat juga dikelompokkan menjadi dua yaitu:

a. *Signature-based Detection*

IDS jenis ini bekerja dengan membandingkan *signature* atau ciri-ciri dari sebuah paket yang telah dianalisis dengan berbagai jenis ciri-ciri serangan yang ada pada databasenya (Prayogo, 2021). Cara kerja dari IDS berbasis ciri-ciri hampir dengan cara kerja aplikasi antivirus dalam melakukan deteksi terhadap malware, dan memiliki satu kekurangan yang cukup signifikan dimana akan terjadi keterlambatan untuk mendeteksi jenis serangan *Zero-day* dikarenakan polanya belum terdapat didatabase. Jenis

serangan *Zero-day* ini sendiri merupakan istilah untuk jenis serangan yang baru diketahui oleh khalayak umum.

b. *Anomaly-based Detection*

IDS jenis ini bekerja dengan mengawasi aktivitas dalam jaringan dan melakukan perbandingan aktivitas jaringan yang terjadi dengan rata-rata aktivitas yang stabil. Sistem akan melakukan mendefinisikan maksud dari jaringan yang normal berdasarkan informasi dari jumlah *bandwith* yang digunakan, *port* yang digunakan untuk terhubung, dan jenis protokol yang digunakan untuk berkomunikasi. Admin akan diberikan peringatan oleh IDS apabila terdapat kejanggalan atau anomali pada aktivitas jaringan yang dipantau (Prayogo, 2021).

2.2 Snort

Snort adalah perangkat lunak untuk mendeteksi segala usaha penyusupan atau serangan terhadap suatu jaringan. Snort merupakan salah satu NIDS *open-source* yang mendeteksi serangan atau penyusupan dengan metode *Signature Detection* yang disimpan pada file *rules*. Struktur suatu *rule* terdiri dari *header* yang akan menentukan tindakan yang akan dilakukan oleh Snort apabila mendeteksi paket yang memiliki kecocokan pola dengan yang didefinisikan juga pada *header* dari *rule* itu. Bagian kedua dari *rule* adalah *option*, dimana penulis dari *rule* dapat memberikan pesan atau deskripsi terhadap pola apa yang dideteksi oleh *rule* ini kepada admin atau operator dari snort (Prayogo, 2021).

Snort merupakan NIDS yang umum diterapkan dan dapat ditemui di berbagai perangkat Firewall sebagai bagian dari solusi IDS perangkat tersebut. Meskipun

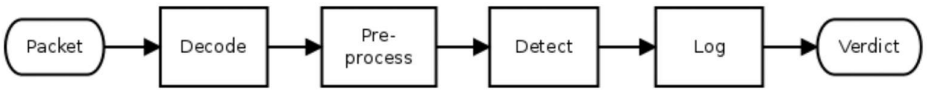
sudah umum digunakan, alasan Snort dijadikan NIDS yang akan dimodifikasi dan dievaluasi karena sumbernya yang umum dan terbuka (Open Source). Popularitas penerapannya di kalangan hobi juga membuat Snort memiliki komunitas yang sangat kuat sehingga referensi terhadap masalah dan/atau tantangan dalam instalasinya relatif mudah untuk ditemukan (*Snort - Network Intrusion Detection & Prevention System*, n.d.).

Snort juga sudah dirilis cukup lama di khalayak publik, dengan rilis pertamanya pada tahun 1998 oleh Martin Roesch yang kemudian mendirikan perusahaan Sourcefire yang akhirnya diakuisisi oleh Cisco pada tahun 2013 menunjukkan tingkat kematangan dari perangkat lunak ini (Rubens, 2015). Dengan sudah lamanya waktu pengembangan dari Snort, sumber kodenya memiliki dokumentasi yang jelas, dan banyak optimasi yang sudah dilakukan terhadap Snort itu sendiri dapat diadaptasikan ke dalam modifikasi yang akan dilakukan untuk memanfaatkan sumber daya GPU sehingga terjadi komputasi secara heterogen (Komputasi dengan dua atau lebih jenis perangkat yang berbeda).

2.2.1 Desain dan Arsitektur Snort

Desain dan Arsitektur Snort berawal dari satu proses master yang akan membuat thread baru untuk melakukan proses terhadap paket yang masuk. Ini merupakan Arsitektur baru dari Snort 3 yang akhirnya memiliki kemampuan multithreading. Dimana thread utama menjadi penghubung antara user dengan aplikasi Snort, mengatur dan mengawasi Input/Output dari aplikasi, dan semua inisialisasi seperti pembacaan file konfigurasi dan rule dari Snort.

Untuk setiap perangkat penghubung jaringan (NIC) yang akan diinspeksi, thread awal akan membuat satu thread analis untuk setiap perangkat itu. Setiap analis akan menerima paket dari NIC untuk kemudian diproses oleh Snort. Adapun tahapan pemrosesan itu ditampilkan di Gambar 3.2 yang diperoleh dari dokumentasi Snort(*Snort 3 User Manual*, n.d.).



Gambar 2.2 Proses Analisis Snort

Pada tahap Decode, analis akan menemukan informasi dasar dari paket yang masuk seperti alamat sumber dan tujuan paket, serta nomor portnya. Selanjutnya dilakukan inspeksi terhadap berbagai protokol enkapsulasi dari paket yang masuk. Setiap paket pada umumnya terdapat beberapa protokol enkapsulasi (eth:ip:tcp:http) sebelum analis bisa melihat pesan atau data dari paket itu. Proses analisa enkapsulasi dilakukan bersamaan dengan proses decoding dari enkapsulasi itu untuk mengetahui apakah ada kejanggalan dari paket yang masuk. Selanjutnya pada tahap Preprocess, analis akan menyiapkan paket yang telah dibongkar pada tahap decode untuk diproses lebih lanjut. Terkadang pada tahap ini dilakukan rekayasa ulang dari paket yang masuk dengan menyusun kembali fragmen IP dan segmen TCP agar terbentuk Protocol Data Unit (PDU) dari paket. PDU kemudian dianalisis dan dinormalisasi sesuai dengan kebutuhan untuk mendukung pemrosesan lebih lanjut. Tahap Detect terbagi atas dua proses. Untuk efisiensi, rule atau aturan yang dibaca oleh Snort untuk menganalisis paket memiliki pola konten spesifik yang dapat dibandingkan dengan paket yang masuk, yang ketika tidak

ditemukan kesamaan, pemrosesan lebih lanjut terhadap paket yang masuk tidak dilanjutkan. Saat inisialisasi awal, aturan-aturan yang dikonfigurasikan dikompilasi menjadi kelompok-kelompok pola. Jika ada satu pola yang memiliki kesamaan dengan paket yang diproses, barulah paket itu dianalisis lebih lanjut. Pada tahap Logging, Snort akan menyimpan hasil dari tahap sebelumnya beserta informasi-informasi terkait sesuai dengan konfigurasi Snort itu sendiri. Umumnya, pada tahap ini juga dilakukan aksi yang menjadi callback apabila suatu kondisi terpenuhi seperti drop atau block paket yang dideteksi terdapat anomali dan/atau kesamaan pola.

Algoritma yang digunakan oleh Snort untuk membandingkan pola yang disimpan dengan paket yang masuk lalu mendeteksi kesamaannya yaitu Aho-Corasick State Machine. Snort menggunakan beberapa penerapan dari Aho-Corasick itu sendiri berdasarkan penggunaan sumber dayanya, yaitu optimasi untuk performansi, atau menghemat penggunaan sumber daya dengan mengorbankan performansi deteksi. Optimasi algoritma Aho-Corasick yang diterapkan Snort menitik beratkan pada besarnya sumber daya memori yang dibutuhkan untuk menyimpan tabel transisi dari mesin otomata Aho-Corasick. Optimasi yang dilakukan terhadap metode penyimpanan tabel transisi terbagi menjadi 4 yaitu Full Matrix, Sparse Matrix, Banded Row, dan Sparse Bands (Norton, n.d.).

Tabel 2.1 Contoh Tabel Transisi Full Matrix

State	Format	Match	2	3	4	5	6	7
0	0	0	0	0	0	0	0	3
1	0	0	0	0	0	4	0	1
2	0	0	4	0	0	0	0	5
3	0	0	0	0	1	0	0	2
4	0	0	0	0	0	2	0	0
5	0	1	0	0	0	0	5	0

Tabel 2.2 Contoh Tabel Transisi Sparse Matrix

State	Format	Match	2	3	4	5	6	7
0	1	0	0	0	0	0	0	3
1	1	0	2	5	4	7	1	
2	1	0	2	2	4	7	5	
3	1	0	2	4	1	7	2	
4	1	0	1	5	2			
5	1	1	1	6	5			

Tabel 2.3 Contoh Tabel Transisi Banded Row

State	Format	Match	2	3	4	5	6	7
0	2	0	1	7	3			
1	2	0	3	5	4	0	1	
2	2	0	4	0	0	0	0	5
3	2	0	4	4	1	0	0	2
4	2	0	1	5	2			
5	2	1	1	6	5			

Tabel 2.4 Contoh Tabel Transisi Sparse Bands

State	Format	Match	2	3	4	5	6	7	8
0	3	0	1	1	7	3			
1	3	0	2	1	5	4	1	7	1
2	3	0	2	1	2	4	1	7	5
3	3	0	2	1	4	1	1	7	2
4	3	0	1	1	5	2			
5	3	1	1	1	6	5			

Dalam penelitian ini, Snort dikonfigurasikan untuk menggunakan format Full Matrix State Machine dimana matriks akan menyimpan seluruh state, termasuk state zero atau failure state dari Automaton Aho-Corasick (Norton, n.d.). Untuk melakukan deteksi pola, Snort akan membentuk beberapa mesin state berdasarkan aturan yang dideklarasikan ketika inisialisasi Snort. Tiap aturan dikelompokkan berdasarkan protokol, port, dan jenis layanannya (SMB, SSH, dll.) dan tiap kelompok akan memiliki mesin state sendiri. Setiap pola yang terdapat dalam aturan ini akan dikompilasi menjadi mesin state.

2.3 Graphics Processing Unit

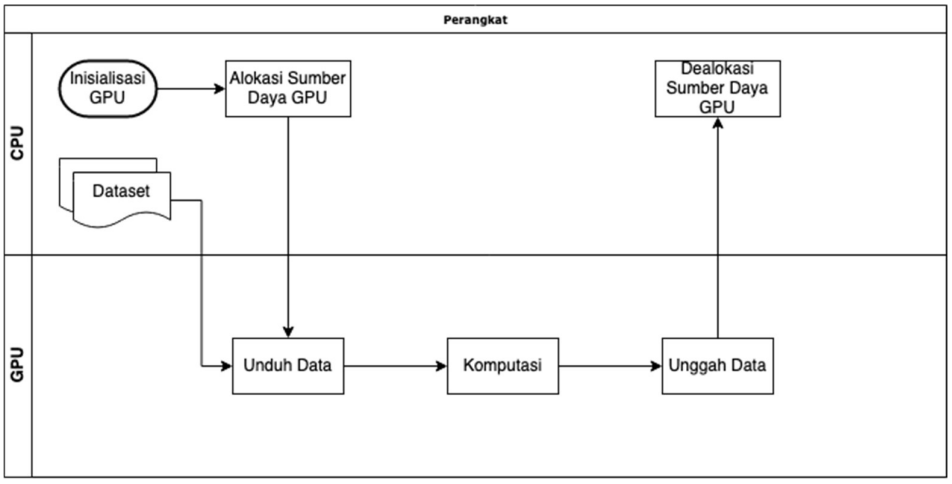
Graphics Processing Unit atau GPU adalah sebuah sirkuit khusus yang dirancang untuk mempercepat *output* gambar dalam *frame buffer* untuk *output* ke layar. GPU sangat efisien dalam memanipulasi grafis komputer dan umumnya lebih efektif daripada CPU. GPU bisa berada pada kartu eksternal (GPU *Card*) atau yang terintegrasi pada CPU. GPU berfungsi untuk mengolah dan memanipulasi grafis pada komputer untuk nantinya ditampilkan pada bentuk visual pada monitor (Wilson et al., 2015).

Seperti halnya CPU, GPU juga membutuhkan memori RAM, BIOS, dan juga I/O sendiri untuk bisa beroperasi, hanya saja karena tugasnya yang lebih spesifik, ia tidak membutuhkan ukuran papan sirkuit sebesar *motherboard* komputer biasanya. Satu perbedaan cara kerja yang paling mendasar dari CPU dan GPU adalah tentang *core* atau otak intinya. CPU yang umumnya digunakan oleh sebagian besar pengguna hanya memiliki tidak lebih dari 8 *core*, sedangkan GPU bisa memiliki ratusan bahkan sampai ribuan *core*. Tetapi *core* yang dimiliki CPU

mempunyai set instruksi yang lebih banyak dari GPU sehingga CPU lebih cocok untuk mengerjakan berbagai macam tugas dalam satu waktu dengan jenis data yang bermacam-macam. Sedangkan GPU lebih cocok untuk tugas yang lebih spesifik dengan input data dengan jenis data yang sama karena kemampuannya untuk melakukan pemrosesan data secara paralel (Santoso, 2019).

2.3.1 General-purpose Computing on Graphics Processing Units

General-purpose Computing on Graphics Processing Units atau GPGPU adalah sebuah teknik komputasi dengan menggunakan GPU untuk melakukan pengolahan data pada aplikasi non-grafis. Cara kerja GPGPU adalah menyalin data pemrosesan dari memori utama ke memori GPU, kemudian GPU akan diberikan instruksi oleh CPU untuk memproses data, dan langkah terakhirnya adalah menyalin data hasil pemrosesan dari memori GPU kembali ke memori utama (Wilson et al., 2015).



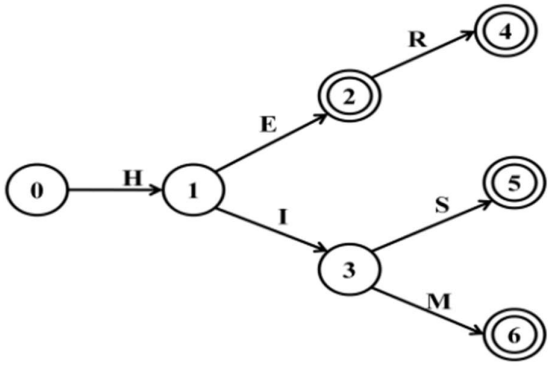
Gambar 2.3 Alur Kerja GPGPU

2.4 Algoritma Aho-Corasick

Algoritma Aho-Corasick merupakan algoritma efisien untuk mendeteksi semua kemunculan dari semua kata kunci dalam suatu kalimat. Algoritma ini terdiri dari pembentukan mesin otomata pencocokan pola dari seluruh kata kunci lalu menggunakan mesin otomata untuk memproses kalimat masukan. Pembentukan mesin otomata membutuhkan waktu yang proporsional dengan jumlah panjang dari semua kata kunci. Jumlah transisi dari mesin otomata yang dibentuk tidak berkaitan dengan jumlah kata kunci. Algoritma ini telah digunakan untuk mempercepat pencarian bibliografi perpustakaan dengan peningkatan performansi 5 sampai 10 kali lebih cepat (Aho & Corasick, 1975).

Algoritma Aho-Corasick adalah salah satu algoritma yang dapat digunakan untuk mencari dimana sebuah string atau disebut dengan pattern, apakah ditemukan di dalam kumpulan string lain dengan ukuran yang lebih besar atau disebut dengan teks. Aho-Corasick adalah algoritma pencocokan pola berbasis kamus yang mencari beberapa set pola pada data masukan, apakah data yang ukurannya lebih kecil atau disebut dengan pola ditemukan dalam kumpulan pola lain yang lebih besar. Pencocokan juga dilakukan secara bersamaan, sehingga kejadian pencocokan dapat ditemukan selama beberapa kali untuk setiap sub pola yang terdapat dalam data input. Algoritma ini melakukan penyusunan *Finite State Machine* (FSM) yang menyerupai pohon dengan beberapa *link* di antara node internalnya. *Link* internal ini memungkinkan perpindahan secara cepat di antara pola yang tidak cocok pada cabang lain yang memiliki prefiks yang sama. Hal ini menyebabkan sistem dapat berpindah secara cepat tanpa perlu melakukan

backtracking atau penelusuran ulang (Situmorang et al., 2018). Pada *Gambar 2.4* merupakan contoh dari *state machine* dengan pola HE, HER, HIS, dan HIM.



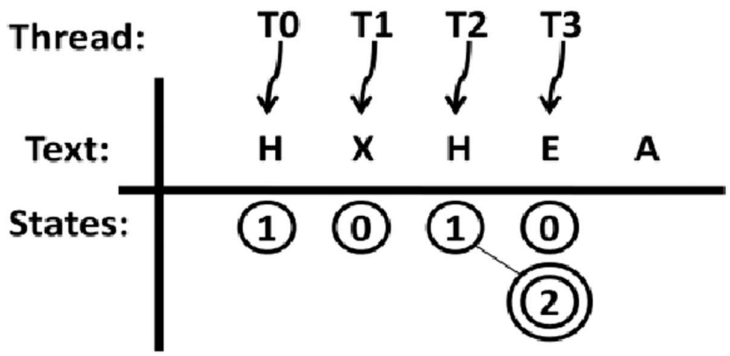
Gambar 2.4 Contoh *State Machine*

2.4.1 Parallel Failureless Aho-Corasick

PFAC atau *Parallel Failureless Aho-Corasick* adalah versi paralel dan merupakan pengembangan dari algoritma Aho-Corasick. Algoritma PFAC tidak akan mengikuti *link* internal atau biasa juga disebut *failure state*, yaitu keadaan dimana pola yang dicari tidak terdapat pada *state machine* dari Aho-Corasick. PFAC akan menggunakan setiap *thread* yang ada untuk membaca *Deterministic Finite State Automata* (DFA) pada memori dan setiap *thread* akan memiliki input *string* masing-masing untuk menelusuri DFA tadi.

Pada PFAC, *state machine* dibuat tanpa mekanisme gagal (*failure state* atau *failure function*). Mekanisme gagal adalah prosedur atau langkah selanjutnya yang akan diambil ketika algoritma gagal menemukan kecocokan pola *input* setelah menelusuri *state machine*, pada PFAC mekanisme ini ditiadakan karena *thread* yang mengalami kegagalan akan langsung di-*terminate* dan *thread* lain akan dipanggil untuk melanjutkan ke *input* selanjutnya. Sebagaimana yang diperlihatkan

pada *Gambar 2.5* dengan *input* HXHEA, dimana T0, T1 dan T2, mengalami terminasi, dan T3 akan terhitung sebagai pola yang ditemukan, karena transisi dari T2 ke T3 berhasil menemukan satu pola yaitu HE. Data kemudian disimpan sesuai dengan metode yang digunakan (Agarwal et al., 2013).



Gambar 2.5 Pola akses PFAC

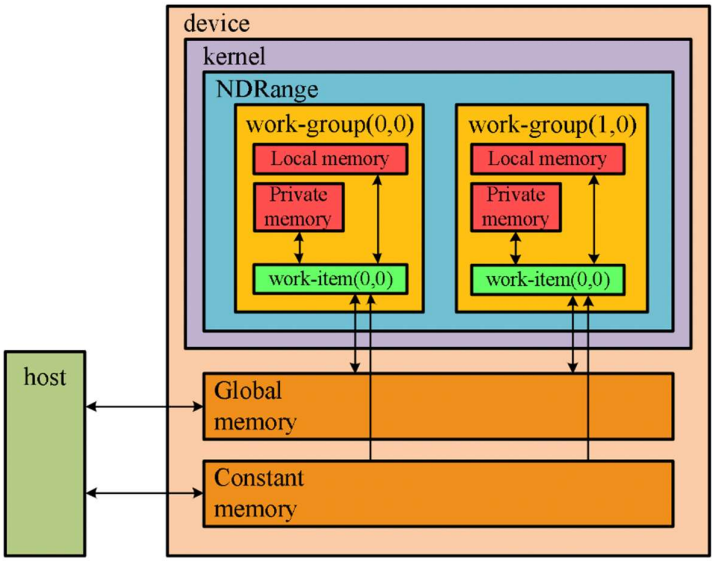
2.5 OpenCL

OpenCL adalah framework untuk pemrograman yang dikelola oleh Khronos Group, yang dapat diimplementasikan pada *heterogeneous platforms* yang terdiri dari Altera, AMD, Apple, ARM Holdings, Creative Technology, IBM, Imagination Technologies, Intel, Nvidia, Qualcomm, Samsung, Vivante, Xilinx, dan ZiiLABS (Imam Cholissodin & Tusty Nadia Maghfira, 2019).

OpenCL adalah kerangka kerja untuk menulis program yang mengeksekusi seluruh platform arsitektur silikon yang terdiri dari CPU / Prosesor dan unit pengolahan grafis / GPU, atau lainnya. OpenCL menyediakan antarmuka standar untuk komputasi paralel berbasis tugas dan paralelisme berbasis data. OpenCL mengambil dan menggabungkan keunggulan dari tiap unit *core*, contoh GPU

memiliki keunggulan dalam komputasi FPU (*Floating Point Unit*). Dengan OpenCL, CPU/Prosesor tidak perlu menghitung bilangan FPU, karena tugas untuk menghitung bilangan FPU telah dialihkan ke GPU. Hal ini akan membuat kinerja CPU lebih ringan dan dapat melakukan tugas lainnya (Imam Cholissodin & Tusty Nadia Maghfira, 2019).

OpenCL di standarisasi oleh Khronos Group dan dipublikasikan pada Desember 2008. Tujuan utama dari OpenCL adalah untuk menyediakan antarmuka aplikasi yang mudah tanpa ada batasan platform. OpenCL menyajikan dua cara untuk melakukan pemrosesan paralel, *Task-parallel* dan *Data-parallel*. Moda *Task-parallel* menggunakan *work-items* untuk mengontrol CPU multi-*core*. Sedangkan mode *Data-parallel* menggunakan ratusan paralel prosesor untuk memproses *work-items* yang sangat banyak. Adapun penerapannya untuk GPGPU, mayoritas aplikasi menggunakan metode *Data-parallel* dari OpenCL (Su et al., 2012).



Gambar 2.6 Arsitektur OpenCL

Pada Gambar 2.6 dapat dilihat diagram arsitektur dari OpenCL, jika program menemukan sebuah fungsi dengan tingkat paralelisme yang tinggi, CPU akan memanggil *kernel* di OpenCL untuk melakukan operasi paralel. Pada *kernel*, unit terkecil dari sebuah komputasi disebut *work-item*. Sebuah kelompok yang terdiri dari beberapa *work-item* disebut dengan *work-group*. Adapun ruang komputasi yang terdiri dari beberapa *work-group* disebut dengan *NDRange*. *Work-item*, *work-group*, dan *NDRange* dapat menentukan dimensi dari ruang komputasi menjadi satu, dua, atau tiga dimensi (Su et al., 2012).

Gambar 2.6 juga memperlihatkan struktur memori dari OpenCL. Terdapat *Private Memory*, *Local Memory*, *Constant Memory*, dan *Global Memory* diurutkan berdasarkan kecepatan aksesnya.

1. *Private Memory*

   Private Memory merupakan memori yang disimpan pada *register* untuk masing-masing *work-item*

2. *Local Memory*

   Local Memory merupakan memori yang terletak pada *compute-unit*, yang bisa diakses oleh semua *work-item* yang tergabung dalam *work-group* yang sama.

3. *Constant Memory*

   Constant Memory merupakan bagian memori dari *device* yang bisa diakses oleh seluruh *work-item* dari *work-group* mana pun dan bersifat *read-only*, tapi dapat dimodifikasi oleh *host*.

4. *Global Memory*

Global Memory merupakan bagian memori dari *device* yang bisa diakses oleh seluruh *work-item* dari *work-group* mana pun dan dapat diakses oleh *host*. Bagian memori ini digunakan untuk transaksi data *host* dan *device*.

2.6 Dataset IoT-23

Dataset IoT-23 adalah kumpulan data baru dari lalu lintas jaringan perangkat IoT. Dataset ini memiliki 20 data rekaman lalu lintas perangkat IoT yang terinfeksi *malware*, dan 3 data rekaman lalu lintas jaringan perangkat IoT biasa. Pertama kali diterbitkan pada Januari 2020, dengan rentang waktu data rekaman mulai dari 2018 hingga 2019. Lalu lintas jaringan IoT ini direkam di Laboratorium *Stratosphere*, grup Kelompok Pusat Kecerdasan Buatan, Fakultas Teknik Elektro, Universitas Teknik Ceko, Republik Ceko. Tujuannya adalah untuk menyajikan data mengenai infeksi *malware* pada perangkat IoT yang nyata dan berlabel, serta lalu lintas normal dari sebuah jaringan perangkat IoT bagi para peneliti untuk mengembangkan algoritma *Machine Learning*. Kumpulan data ini dan penelitiannya didanai oleh Avast Software, Praha (Parmisano et al., 2020).

Pada data yang terinfeksi *malware*, sebuah sampel dijalankan pada sebuah Raspberry Pi, yang menggunakan beberapa protocol dan melakukan tindakan yang berbeda. Tabel 2.5 menunjukkan data-data yang berisi rekaman lalu lintas jaringan perangkat IoT yang terinfeksi *malware*.

Tabel 2.5 Data Lalu Lintas Jaringan Perangkat IoT yang Terinfeksi

#	Name of Dataset	Duration (hrs)	#Packets	#ZeekFlows	Pcap Size	Name
1	CTU-IoT-Malware-Capture-34-1	24	233,000	23,146	121 MB	Mirai
2	CTU-IoT-Malware-Capture-43-1	1	82,000,000	67,321,810	6 GB	Mirai
3	CTU-IoT-Malware-Capture-44-1	2	1,309,000	238	1.7 GB	Mirai
4	CTU-IoT-Malware-Capture-49-1	8	18,000,000	5,410,562	1.3 GB	Mirai
5	CTU-IoT-Malware-Capture-52-1	24	64,000,000	19,781,379	4.6 GB	Mirai
6	CTU-IoT-Malware-Capture-20-1	24	50,000	3,210	3.9 MB	Torii
7	CTU-IoT-Malware-Capture-21-1	24	50,000	3,287	3.9 MB	Torii
8	CTU-IoT-Malware-Capture-42-1	8	24,000	4,427	2.8 MB	Trojan
9	CTU-IoT-Malware-Capture-60-1	24	271,000,000	3,581,029	21 GB	Gagfyt
10	CTU-IoT-Malware-Capture-17-1	24	109,000,000	54,659,864	7.8 GB	Kenjiro
11	CTU-IoT-Malware-Capture-36-1	24	13,000,000	13,645,107	992 MB	Okiru
12	CTU-IoT-Malware-Capture-33-1	24	54,000,000	54,454,592	3.9 GB	Kenjiro
13	CTU-IoT-Malware-Capture-8-1	24	23,000	10,404	2.1 MB	Hakai
14	CTU-IoT-Malware-Capture-35-1	24	46,000,000	10,447,796	3.6G	Mirai
15	CTU-IoT-Malware-Capture-48-1	24	13,000,000	3,394,347	1.2G	Mirai
16	CTU-IoT-Malware-Capture-39-1	7	73,000,000	73,568,982	5.3GB	IRCBot
17	CTU-IoT-Malware-Capture-7-1	24	11,000,000	11,454,723	897 MB	Linux,Mirai
18	CTU-IoT-Malware-Capture-9-1	24	6,437,000	6,378,294	472 MB	Linux.Hajime
19	CTU-IoT-Malware-Capture-3-1	36	496,000	156,104	56 MB	Muhstik
20	CTU-IoT-Malware-Capture-1-1	112	1,686,000	1,008,749	140 MB	Hide and Seek

Data lalu lintas jaringan normal dari perangkat IoT diperoleh dengan merekam lalu lintas jaringan dari tiga perangkat IoT yang berbeda yaitu Philips Hue Smart LED Lamp, Amazon Echo, dan Somfy Smart Doorlock. Tabel 2.6 menunjukkan data dari lalu lintas jaringan dari masing-masing perangkat yang digunakan.

Tabel 2.6 Data Lalu Lintas Jaringan Perangkat IoT Normal

#	Name of Dataset	Duration(~hrs)	#Packets	#ZeekFlows	Pcap Size	Device
21	CTU-Honeypot-Capture-7-1	1.4	8,276	139	2,094 KB	Somfy Door Lock
22	CTU-Honeypot-Capture-4-1	24	21,000	461	4,594 KB	Philips HUE
23	CTU-Honeypot-Capture-5-1	5.4	398,000	1,383	381 MB	Amazon Echo

2.7 Basic Instrumentation Profiler

Basic Instrumentation Profiler adalah metode *profiling* yang dipublikasikan oleh Yan Chernikov, merupakan sebuah metode *profiling* fungsi dan/atau baris dari sebuah program C++. Metode ini mengukur lamanya suatu fungsi berjalan dengan mencatat waktu mulai ketika objek *Instrumentation Timer* dibuat, lalu mencatat waktu berhenti ketika objek *Instrumentation Timer* dihancurkan, atau dengan memanggil fungsi **Stop()** pada objek *Instrumentation Timer* yang didefinisikan sebelumnya (Chernikov, 2019). Luaran dari metode ini kemudian diproses dan divisualisasikan menggunakan Perfetto (*Perfetto - System Profiling, App Tracing and Trace Analysis - Perfetto Tracing Docs*, n.d.).