HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Quốc Hữu

NGHIÊN CỨU XÂY DỰNG GIẢI PHÁP PHÂN TÁN ĐỘNG TRONG HỆ THỐNG PHÂN TÍCH MÃ ĐỘC IOT BOTNET DỰA TRÊN KAFKA VÀ KSQL

LUẬN VĂN THẠC SĨ KỸ THUẬT

(*Theo định hướng ứng dụng*)

HÀ NỘI - NĂM 2022

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Quốc Hữu

NGHIÊN CỨU XÂY DỰNG GIẢI PHÁP PHÂN TÁN ĐỘNG TRONG HỆ THỐNG PHÂN TÍCH MÃ ĐỘC IOT BOTNET DỰA TRÊN KAFKA VÀ KSQL

Chuyên ngành: HỆ THỐNG THÔNG TIN
Mã số: 8.48.01.04 (Hệ thống thông tin)

LUẬN VĂN THẠC SĨ KỸ THUẬT
(*Theo định hướng ứng dụng*)

NGƯỜI HƯỚNG DẪN KHOA HỌC:
T.S NGÔ QUỐC DŨNG

HÀ NỘI - NĂM 2022

LỜI CAM ĐOAN

Tôi cam đoan luận văn với đề tài *"Nghiên cứu xây dựng giải pháp phân tán động trong hệ thống phân tích mã độc IoT botnet dựa trên Kafka và KSQL"* là công trình nghiên cứu của riêng tôi.

Các số liệu, kết quả nêu trong luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tác giả luận văn ký và ghi rõ họ tên

LỜI CẢM ƠN

Trong quá trình nghiên cứu, tìm hiểu và thực hiện luận văn này, học viên đã nhận được sự giúp đỡ rất nhiệt tình của TS. Ngô Quốc Dũng. Học viên xin chân thành cám ơn thầy đã bỏ công sức giúp định hướng, thực hiện và hoàn thành được luận văn này.

Học viên xin chân thành cám ơn các thầy/cô khoa Sau đại học và các thầy/cô khoa Công Nghệ Thông Tin 1-Học Viện Công Nghệ Bưu Chính Viễn Thông đã đào tạo về kiến thức, văn hóa cho học viên, cung cấp các nền tảng để áp dụng hoàn thành luận văn này.

Cuối cùng, học viên xin cảm ơn gia đình và bạn bè tạo điều kiện thuận lợi, giúp đỡ học viên trong suốt quá trình học tập tại Học Viện Công Nghệ Bưu Chính Viễn Thông cũng như quá trình nghiên cứu luận văn này.

Học viên xin chân thành cám ơn!

Hà Nội, ngày tháng năm

Học viên

Nguyễn Quốc Hữu

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, THUẬT NGỮ VIẾT TẮT

Thuật ngữ	Tiếng Anh	Tiếng Việt
ACO	Ant colony optimization	Giải thuật tối ưu hoá đàn kiến
AI	Artificial intelligence	Trí tuệ nhân tạo
ASG	Adversarial Samples Generator	Bộ tạo mẫu đối nghịch
C&C	Control and Command	Máy chủ điều khiển và thực thi mã độc Botnet
CPU	Central Processing Unit	Bộ vi xử lý trung tâm
CSS	Cascading Style Sheets	Ngôn ngữ lập trình CSS
DDOS	Distributed Denial of service	Tấn công từ chối dịch vụ
DLL	Dynamic Link Library	Thư viện liên kết động Windows
DNS	Domain Name System	Hệ thống phân giải tên miền
DT	Decision tree	Cây quyết định
ELF	Extensible Linking Format	Định dạng liên kết thực thi
EPRS	European Parliamentary Research Service	Dịch vụ Nghiên cứu Nghị viện Châu Âu
FVE	Feature Vectors Extractor	Bộ trích xuất véc tơ đặc trưng
HTTP	Hyper Text Transfer Protocol	Giao thức Truyền tải Siêu Văn Bản
IBD	IoT Botnet Dectector	Bộ phát hiện IoT Botnet
IoT	Internet of Things	Internet vạn vật
IP	Internet Protocol	Giao thức Internet
IPC	Inter Process Communuication	Giao tiếp liên tiến trình
IPv4	Internet Protocol version 4	Giao thức Internet phiên bản 4
IRC	Internet Relay Chat	Trò chuyện chuyển tiếp Internet
ITU	International Telecommunication Union	Liên minh Viễn thông thế giới
JSON	JavaScript Object Notation	Định dạng dữ liệu JSON
KNN	K nearest neighbor	Thuật toán K nearest neighbor
MD5	Message-Digest algorithm 5	Thuật toán Tiêu hóa-tin nhắn 5
NB	Naive Bayes	Thuật toán học máy Naïve Bayes
PE	Portable Executable	Định dạng tệp cho tệp thực thi

PID	Proccess ID	Định danh tiến trình
PSI	Printable Strings Information	Thông tin chuỗi
QoS	Quality of service	Chất lượng dịch vụ
RAM	Random Access Memory	Bộ nhớ truy xuất ngẫu nhiên
RF	Random Forest	Thuật toán rừng ngẫu nhiên
RFID	Radio Frequency Indentification	Nhận dạng qua tần số vô tuyến)
RL	Reinforcement learning	Học tăng cường
RLA	Reinforcement learning Agent	Tác từ học tăng cường
SCC	System Calls Collector	Bộ thu thập lệnh gọi hệ thống
SCG	System call graph	Đồ thị lệnh gọi hệ thống
SHA-1	Secure Hash Algorithm 1	Thuật toán băm an toàn 1
SSH	Secure Socket Shell	Giao thức SSH
SVM	Support vector machine	Máy vectơ hỗ trợ
TCP	Transmission Control Protocol	Giao thức điều khiển truyền vận
URL	Uniform Resource Locator	Hệ thống định vị tài nguyên thống nhất
API	Application Programming Interface	Giao diện lập trình ứng dụng
SQL	Structured Query Language	Ngôn ngữ truy vấn mang tính cấu trúc
ML	Machine learning	Học máy
GSLB	Global server load balancing	Cân bằng tải máy chủ toàn cầu

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH

MỞ ĐẦU

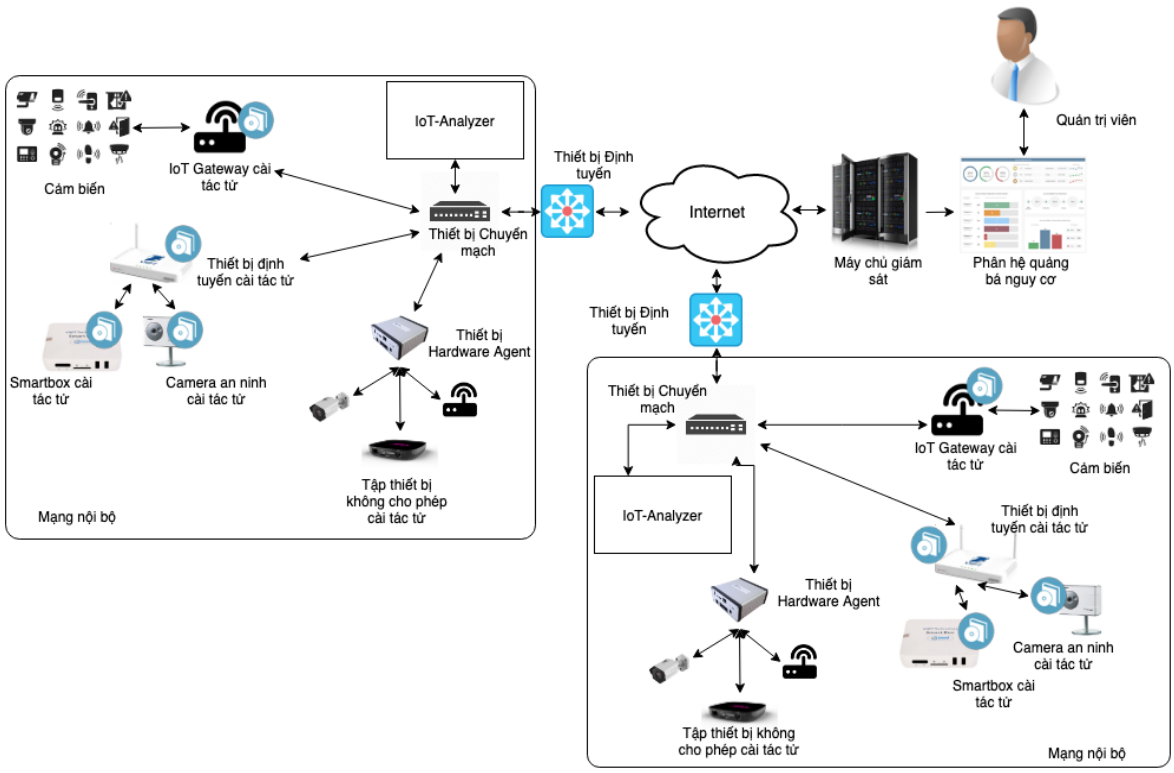
1. Lý do chọn đề tài.

Cách mạng công nghiệp 4.0 mang đến rất nhiều những thay đổi nhanh chóng về công nghiệp trên toàn thế giới, đặc biệt là mở ra các công nghệ của tương lai. Một trong số các công nghệ được phát triển rất mạnh trong cách mạng công nghiệp 4.0 là Internet vạn vật (Internet of thing – IoT). Mạng lưới vạn vật kết nối Internet hoặc là mạng lưới thiết bị kết nối Internet viết tắt là IoT là một kịch bản của thế giới, khi mà mỗi đồ vật, con người được cung cấp một định danh của riêng mình, và tất cả có khả năng truyền tải, trao đổi thông tin, dữ liệu qua một mạng duy nhất mà không cần đến sự tương tác trực tiếp giữa người với người, hay người với máy tính. IoT đã phát triển từ sự hội tụ của công nghệ không dây, công nghệ vi cơ điện tử và Internet. Nói đơn giản là một tập hợp các thiết bị có khả năng kết nối với nhau, với Internet và với thế giới bên ngoài để thực hiện một công việc nào đó. Song song với sự phát triển rất nhanh về mặt công nghệ, các thiết bị IoT cũng có có rất nhiều các nhược điểm cần khắc phục, một trong số các nhược điểm rất lớn của thiết bị IoT là bảo mật. Các thiết bị IoT có đặc điểm là hạn chế về tài nguyên, năng lực xử lý thấp và bộ nhớ nhỏ. Bởi vì thiết bị IoT phải đáp ứng với rất nhiều loại môi trường khác nhau, điều này làm cho thiết bị IoT phải đối mặt với thách thức về phát triển các giải pháp bảo mật. Sự thiếu hụt các cơ chế và tiêu chuẩn bảo mật dẫn đến rất nhiều lỗ hổng trên thiết bị IoT được khai thác. Khi kẻ tấn công kiểm soát được một số lượng lớn thiết bị IoT sẽ rất nguy hiểm bởi có thể gây ra những cuộc tấn công lớn và cực lớn. Một trong số các kịch bản tấn công phổ biến là kẻ tấn công sẽ sử dụng các thiết bị IoT kiểm soát được trở thành một phần của một IoT botnet. Bằng cách đó, kẻ tấn công có thể tạo ra và mở rộng IoT botnet, sau đó sử dụng trong các cuộc tấn công trên không gian mạng khác nhau. Các cuộc tấn công của IoT botnet có thể gây ra hậu quả rất nghiêm trọng. Việc đảm bảo an toàn thông tin cho các thiết bị IoT đã và đang là một chủ đề được nghiên cứu rất nhiều trên thế giới.

Hệ thống phát hiện và cảnh báo mã độc IoT Botnet đã và đang được phát triển bởi các nhà nghiên cứu cũng như các hãng sản xuất phần mềm/phần cứng trên thế

giới. Trong đó TS. Ngô Quốc Dũng đã đưa ra một giải pháp hệ thống gồm tiền xử lý, xử lý, phát hiện cảnh báo và ngăn chặn các cuộc tấn công mạng nhằm vào các thiết bi IoT cỡ nhỏ phù hợp với các thiết bị IoT phổ biến tại Việt Nam. Mô hình của hệ thống được mô tả trong Hình 1. Từ những dữ liệu thu được tại các thiết bị IoT, phân hệ xử lý và phân tích dữ liệu (IoT-Analyzer) tiến hành phân tích và cảnh báo các nguy cơ an ninh mạng trước khi gửi về máy chủ giám sát tập trung. Mỗi điểm phân tích này gồm 2 mô đun chính:

+ Mô đun tiền xử lý và chuẩn hoá dữ liệu thu thập được từ các thiết bị dưới sự quản lý của từng điểm trong hệ thống;
+ Mô đun áp dụng mô hình học máy trong việc phân tích và phát hiện các nguy cơ tấn công mạng.



Mô hình tổng quan hệ thống tự động phát hiện, cảnh báo và ngăn chặn tấn công mạng nhằm vào các thiết bị IoT cỡ nhỏ sử dụng mạng lưới tác tử thông minh.

Bài toán đặt ra khi triển khai hệ thống là tại một mạng nội bộ, nếu một IoT-Analyzer bị quá tải thì mạng nội bộ nó chịu trách nhiệm sẽ có nguy cơ mất an toàn

bảo mật. Do đó cần có cơ chế xử lý khi một IoT-Analyzer quá tải và đưa ra hướng giải quyết dựa vào các IoT-Analyzer khác không bị quá tải. Bằng hướng nghiên cứu này, học viên đã chọn đề tài "***Nghiên cứu xây dựng giải pháp phân tán động trong hệ thống phân tích mã động IoT Botnet dựa trên KAFKA và KSQL***" nhằm cải thiện nhược điểm của hệ thống.

2. Tổng quan về đề tài nghiên cứu.

Hiện nay, có rất nhiều các nghiên cứu và khảo sát về hệ thống xử lý phân tán nhằm tang hiệu quả của việc xử lý dữ liệu. HARUNA ISAH và cộng sự [1] đã trình bày nghiên cứu so sánh về các khung phân tích và xử lý luồng dữ liệu phân tán và đánh giá quan trọng về các khung xử lý luồng dữ liệu phân tán mã nguồn mở đại diện (Storm, Spark Streaming, Flink, Kafka Streams) và mã nguồn thương mại (IBM Streams). Nghiên cứu cũng báo cáo về kiến trúc phân tích đa luồng có thể đóng vai trò là hướng dẫn cho các tổ chức và cá nhân có kế hoạch triển khai khung phân tích và xử lý luồng dữ liệu theo thời gian thực. Supun Kamburugamuve và Geoffrey Fox [2] đã đưa ra đánh giá một hệ thống xử lý luồng trực tuyến trên hai khía cạnh độc lập. Khía cạnh thứ nhất, có một API lập trình để phát triển các ứng dụng xử lý luồng trực tuyến và khía cạnh còn lại là có một công cụ thực thi thực thi ứng dụng xử lý luồng trực tuyến.

Các chương trình mã nguồn mở phổ biến để xử lý luồng dữ liệu trên thế giới như Apache Spark, Apache Flink, Apache Kafka. Apache Kafka là một nền tảng xử lý luồng trực tuyến phân tán. Đối với các thay đổi phức tạp, Kafka cung cấp API STREAMS được tích hợp đầy đủ. Api Streams cho phép ứng dụng hoạt động như một bộ xử lý luồng, sử dụng luồng đầu vào từ một hoặc nhiều chủ đề và tạo luồng đầu ra thành một hoặc nhiều chủ đề đầu ra, chuyển luồng đầu vào thành luồng đầu ra một cách hiệu quả. Kafka hỗ trợ cho nhiều kiểu xử lý luồng, giúp giải quyết các vấn đề khó khăn mà ứng dụng gặp phải: xử lý dữ liệu không theo thứ tự, xử lý lại đầu vào khi thay đổi mã, thực hiện tính toán trạng thái, v.v. KSQL là công cụ xử lý luồng cho phép xử lý dữ liệu thời gian thực dựa trên Apache Kafka. Nó cung cấp một giao diện SQL tương tác mạnh mẽ, dễ sử dụng để xử lý luồng trên Kafka mà không cần phải

viết mã bằng ngôn ngữ lập trình như Java hoặc Python. KSQL có khả năng mở rộng, đàn hồi, chịu được lỗi và hỗ trợ một loạt các hoạt động xử lý luồng trực tuyến, bao gồm lọc dữ liệu, chuyển đổi, tổng hợp, nối, cửa sổ và phiên.

Để giải quyết bài toán đã nêu trong mục 1, mục tiêu của học viên là nghiên cứu và đề xuất mô hình xử lý phân tán động trong hệ thống phân tích mã độc IoT Botnet dựa trên nền tảng các công cụ Kafka, KSQL…

3. Mục đích nghiên cứu.

Nghiên cứu và xây dựng giải pháp phân tán động dựa trên nền tảng các công nghệ mới, cụ thể là Kafka và KSQL nhằm tối ưu hoá khả năng phân tích và phát hiện mã độc IoT Botnet.

4. Đối tượng và phạm vi nghiên cứu.

- Đối tượng nghiên cứu: Hệ thống xử lý phân tán động trên nền tảng công nghệ Kafka, KSQL…

- Phạm vi nghiên cứu: Hệ thống phân tích mã độc IoT Botnet.

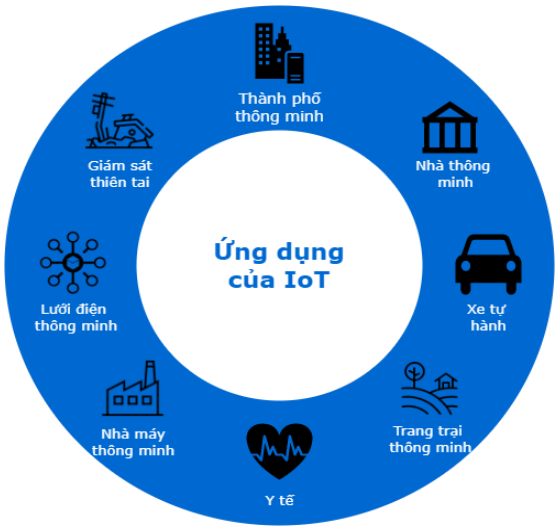
5. Phương pháp nghiên cứu.

- Phương pháp nghiên cứu lý thuyết: Phân tích và tổng hợp các tài liệu liên quan đến đề tài. Phân loại và hệ thống hoá lý thuyết các kết quả nghiên cứu hiện tại như bài báo, kết quả nghiên cứu khoa học về phân tán động, phân tích và phát hiện mã độc IoT botnet.

- Phương pháp nghiên cứu thực tiễn: Sử dụng phương pháp thực nghiệm khoa học để xây dựng, đánh giá giải pháp phân tán động.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

Trong sự phát triển mạnh mẽ của nền công nghiệp 4.0, internet vạn vật là một lĩnh vực đã và đang được nghiên cứu và ứng dụng rộng rãi. Trong những năm gần đây, sự bùng nổ về số lượng của thiết bị IoT giúp con người rất nhiều trong đời sống cũng như nhiều ngành công nghiệp. Tuy nhiên bên cạnh ưu điểm vượt trội về ứng dụng, các thiết bị IoT cũng có các đặc điểm liên quan đến bảo mật mà hacker có thể khai thác. Với số lượng thiết bị IoT tăng trưởng từng ngày, việc nghiên cứu các biện pháp bảo mật cho thiết bị IoT đang rất được quan tâm hiện nay. Ngoài việc xây dựng hệ thống phân tích và phát hiện mã độc, tối ưu hiệu năng hệ thống để có thể phát hiện sớm mã độc cũng là một công việc rất quan trọng. Trong chương này, luận văn sẽ trình bày tổng quan về mã độc IoT botnet và mô hình hệ thống phân tích, phát hiện mã độc IoT botnet.



Hình 1.1: Ứng dụng của IoT trong thực tế

1.1 Tổng quan mã độc IoT Botnet

1.1.1 Các khái niệm trong IoT

Internet of Things là một ý tưởng đã được đưa ra từ lâu, tuy nhiên cho đến năm 1999 mới chính thức được đưa ra bởi Kevin Ashton [1] – Giám đốc điều hành của viện Auto-ID của Viện công nghệ Massachusets (MIT - Massachusetts Institute of Technology). Trong đó các thiết bị được trang bị Radio Frequency Indentification (RFID) là điều kiện tiên quyết cho thiết bị IoT vào thời điểm đó. Cụm từ Internet of Things để chỉ các đối tượng có thể gán nhãn (tagged), quản lý và theo dõi.

Hiện nay khái niệm về IoT có rất nhiều cách giải thích khác nhau được các tổ chức đưa ra, một trong các tổ chức uy tín trên thế giới là Liên minh Viễn thông thế giới (ITU – International Telecommunication Union) [2] đã đưa ra một khái niệm tương đối hoàn chỉnh về IoT. Khái niệm IoT theo ITU được phát biểu như sau: "*Internet of things là một cơ sở hạ tầng mang tính toàn cầu cho xã hội thông tin, mang đến những dịch vụ tiên tiến bằng cách kết nối các "đồ vật" (cả vật lý lẫn ảo) dựa trên sự tồn tại của thông tin, dựa trên khả năng tương tác của các thông tin đó và dựa trên các công nghệ truyền thông. Thông qua việc khai thác khả năng nhận biết, thu thập xử lý dữ liệu, công nghệ các hệ thống IoT tận dụng mọi thứ để cung cấp dịch vụ cho tất cả các loại ứng dụng khác nhau, đồng thời bảo đảm tính bảo mật và quyền riêng tư"*. Nhìn chung, các khái niệm về IoT có thể chưa đồng bộ trên thế giới, tuy nhiên về cơ bản là xoay quanh các thiết bị có kết nối với nhau để thực hiện một mục đích nhất định cho con người.

Từ định nghĩa về IoT nói chung chỉ ra một mạng lưới các vật (things) kết nối và tương tác với nhau, vậy các vật (things) là gì và cụ thể được ứng dụng như thế nào trong đời sống và các lĩnh vực khác? Trong cụm từ 'IoT', từ 'thing' biểu thị một thực thể vật lý hoặc ảo hoá có chức năng nhất định. Đây có thể là thiết bị gia dụng, công nghệ đeo được, hệ thống an ninh hoặc các thiết bị được kết nối/kết nối khác. Trong môi trường ứng dụng của IoT, các thiết bị IoT đóng vai trò là thực thể thiết yếu và đa kiến trúc. Từ đó, trong khuôn khổ luận văn và phù hợp với nội dung nghiên cứu, học viên đưa ra khái niệm thiết bị IoT như sau:

Định nghĩa 1: Thiết bị IoT là những thực thể vật lý hoặc ảo hoá đa kiến trúc, có các đặc điểm cụ thể như tài nguyên hạn chế, nhỏ gọn để phục vụ mục các đích khác nhau, có khả năng xử lý dữ liệu, kết nối và chia sẻ thông tin với các thực thể khác.

Trong thực tế, có rất nhiều loại thiết bị IoT điển hình như camera, thiết bị định tuyến, ti vi, máy in, cảm biến, đồng hồ, điện thoại, …. Thiết bị IoT được áp dụng trong đa số các lĩnh vực hiện nay và không còn quá xa lạ tại Việt Nam. Tiềm năng của IoT trên thế giới là rất lớn và đang được nghiên cứu ứng dụng rộng rãi như nhà thông minh, thành phố thông minh, trang trại thông minh, nhà máy thông minh, lưới điện thông minh, y tế. Các thiết bị IoT hiện nay chưa có một tiêu chuẩn nào để sản

xuất và phát triển, vì vậy thiết bị IoT có đa kiến trúc vi xử lý, đa hệ điều hành và phổ biến nhất là các biến thể của hệ điều hành Linux.

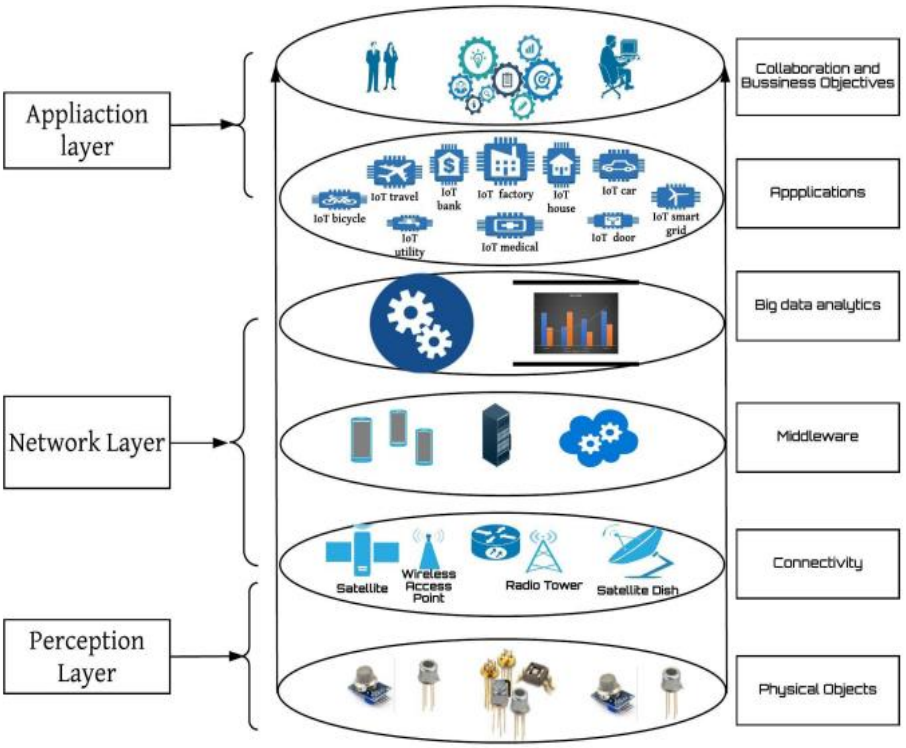
a) Kiến trúc của IoT

Kể từ khi những nghiên cứu đầu tiên về IoT được thực hiện, kiến trúc ba lớp đã là mô hình tổng quan cho các ứng dụng IoT. Ba lớp là Nhận thức (hoặc Thiết bị), Mạng và Ứng dụng.

Lớp nhận thức: Bản thân các cảm biến nằm trên lớp này. Đây là nơi bắt nguồn của dữ liệu. Dữ liệu có thể được thu thập từ bất kỳ số lượng cảm biến nào trên thiết bị được kết nối. Các thiết bị truyền động, hoạt động dựa trên môi trường của chúng, cũng nằm ở lớp này của kiến trúc.

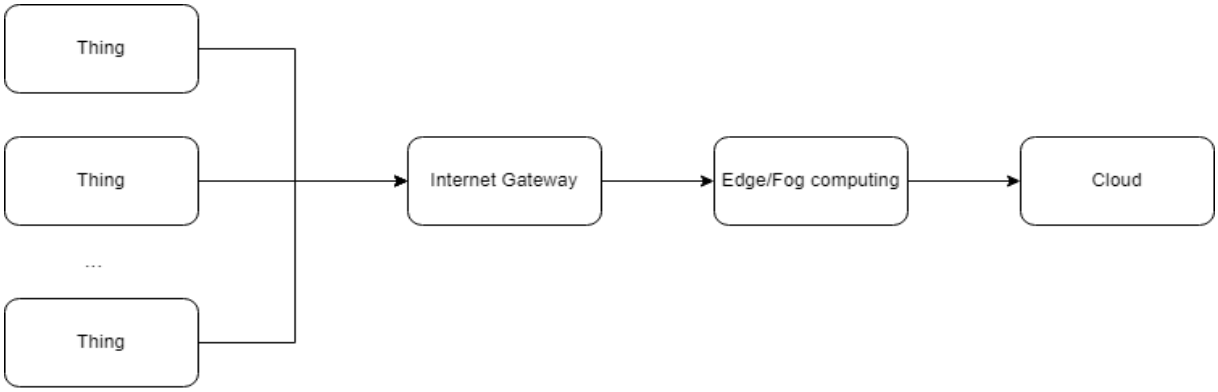
Lớp mạng: Lớp mạng mô tả lượng lớn dữ liệu đang di chuyển trong ứng dụng. Lớp này kết nối các thiết bị khác nhau và gửi dữ liệu đến các dịch vụ back-end thích hợp.

Lớp ứng dụng: Lớp ứng dụng là những gì người dùng nhìn thấy. Đây có thể là một ứng dụng để điều khiển một thiết bị trong hệ sinh thái nhà thông minh hoặc một bảng điều khiển hiển thị trạng thái của các thiết bị là một phần của hệ thống.



Hình 1.2: Các lớp trong kiến trúc IoT

Kiến trúc Internet of Things được thể hiện trong hình 1.3 sử dụng cách tiếp cận bốn giai đoạn. Kiến trúc này mô tả các khối xây dựng khác nhau tạo thành giải pháp IoT. Vật (thing) có thể là bất kỳ thiết bị, dữ liệu hoặc đối tượng thông minh nào và tất cả những thứ này đều được kết nối với đám mây thông qua một cổng chủ yếu là internet. Các thành phần riêng lẻ như như quản lý thiết bị, phân tích, trực quan hóa, v.v. được sử dụng để quản lý dữ liệu với sự trợ giúp của Dữ liệu lớn được sử dụng để lưu trữ dữ liệu.



Hình 1.3: Kiến trúc của hệ thống IoT theo bốn giai đoạn

Thiết bị (Device/Thing): Giai đoạn này là về các thiết bị thực tế trong các giải pháp IoT. Các thiết bị này có thể là cảm biến hoặc thiết bị truyền động trong lớp nhận thức (Perceptron). Các thiết bị đó sẽ tạo ra dữ liệu (trong trường hợp cảm biến) hoặc hoạt động trên môi trường của chúng (trong trường hợp bộ truyền động). Dữ liệu được tạo ra được chuyển đổi dưới dạng kỹ thuật số và truyền đến cổng kết nối internet. Trừ khi phải đưa ra quyết định quan trọng, dữ liệu thường được gửi ở trạng thái thô đến giai đoạn tiếp theo do tài nguyên của chính thiết bị có hạn.

Cổng Internet (Internet Gateway): Cổng Internet sẽ nhận dữ liệu thô từ thiết bị và xử lý trước khi gửi lên đám mây. Cổng kết nối internet này có thể được gắn vật lý vào thiết bị hoặc một thiết bị độc lập có thể giao tiếp với các cảm biến qua mạng công suất thấp và chuyển tiếp dữ liệu đến internet.

Điện toán biên hoặc sương mù (Edge/Fog computing): Để xử lý dữ liệu nhanh nhất có thể mà không cần đến các thực thể trong điện toán đám mây. Điều này sẽ cho phép phân tích dữ liệu nhanh chóng và xác định xem có điều gì cần chú ý ngay lập tức hay không. Lớp này thường chỉ quan tâm đến dữ liệu gần đây được yêu cầu cho

các hoạt động quan trọng về thời gian. Một số quá trình xử lý trước cũng có thể được thực hiện ở giai đoạn này, để hạn chế dữ liệu cuối cùng được chuyển lên đám mây.

Đám mây hoặc trung tâm dữ liệu (Cloud or datacenter): Trong giai đoạn cuối cùng này, dữ liệu được lưu trữ để xử lý sau. Các lớp ứng dụng và nghiệp vụ nằm trong giai đoạn này, nơi các phần mềm quản lý có thể được cung cấp thông qua dữ liệu được lưu trữ trên đám mây. Phân tích sâu hoặc các hoạt động sử dụng nhiều tài nguyên như đào tạo máy học sẽ xảy ra ở giai đoạn này.

b) Đặc điểm của thiết bị IoT

Thiết bị IoT có các ứng dụng khác nhau, nên chúng cũng sẽ có rất nhiều đặc điểm khác nhau. Tuy nhiên các thiết bị IoT có một số đặc điểm chung được trình bày trong phần này. Khác với những thiết bị trên Internet thông thường, thiết bị IoT có các đặc điểm như sau:

Tính không đồng nhất: Các thiết bị trong IoT là không đồng nhất vì dựa trên rất nhiều nền tảng phần cứng và mạng. Các thiết bị này có thể trao đổi, tương tác thông tin với thiết bị khác hoặc các nền tảng dịch vụ khác thông qua các mạng khác nhau. Do dịch vụ và mục đích sản xuất khác nhau, các nhà sản xuất sử dụng các phần cứng khác nhau. Các vi xử lý được các nhà sản xuất phần cứng tối ưu khác so với máy tính cá nhân, các thiết bị IoT sử dụng các kiến trúc vi xử lý cỡ nhỏ như: MIPS, ARM, PowerPC, MIPSEL, … Các giao thức kết nối trong mạng IoT cũng được sử dụng khác nhau như: mạng di động, mạng diện rộng công suất thấp (Low Power Wide Area Networks), mạng Zigbee và các giao thức lưới, Bluetooth, Wi-Fi, RFID, …

Tài nguyên hạn chế và thiết bị nhỏ gọn: Nhằm mục đích tối thiểu hoá chi phí sản xuất và làm cho thiết bị có kích thước nhỏ gọn, các phần cứng không cần thiết trong các thiết bị IoT được loại bỏ một cách triệt để. Do đó các thiết bị IoT bị hạn chế rất nhiều về tài nguyên như bộ nhớ thấp, năng lực tính toán thấp.

Thay đổi động: Các trạng thái của thiết bị IoT có thể thay đổi động về các trạng thái như ngủ, thức, kết nối, ngắt kết nối dựa vào các bối cảnh làm việc cũng như vị trí, vận tốc của thiết bị. Thêm nữa, số lượng thiết bị trong mạng cũng luôn luôn cần phải thay đổi một cách linh hoạt.

Quy mô mở rộng lớn: Số lượng các thiết bị IoT ngày càng được mở rộng và tăng lên một cách nhanh chóng. Số lượng này lớn hơn số lượng các máy tính sử dụng Internet rất nhiều. Trong báo cáo của Cisco [3], đến năm 2023 sẽ có tới 29 tỷ thiết bị IoT trên toàn cầu.

Tính kết nối liên thông: Với các hệ thống IoT, tất cả mọi vật được kết nối liên thông với thông tin toàn cầu và hạ tầng liên lạc.

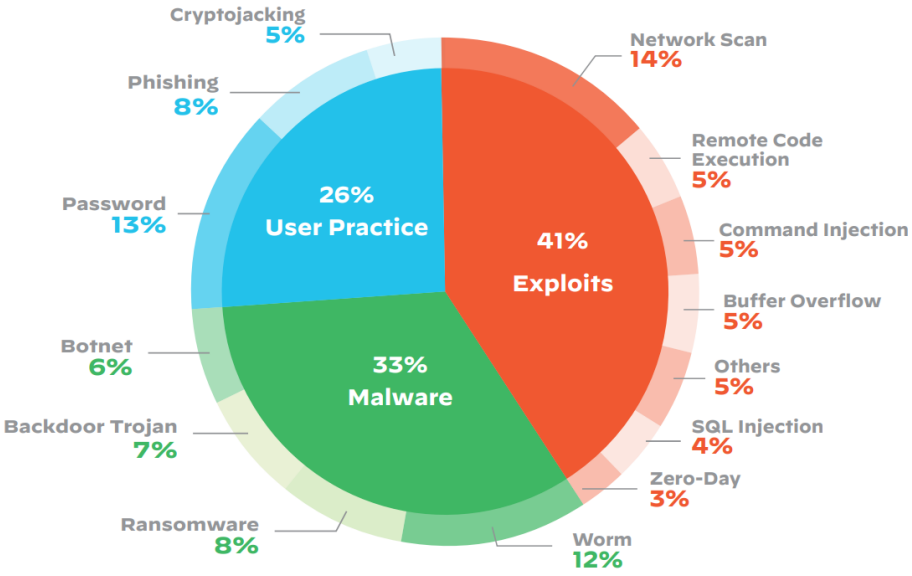
1.1.2 Mã độc IoT Botnet

Như đã trình bày ở phần trên, dựa vào đặc điểm thiết bị IoT có khả năng bị khai thác và tấn công là rất lớn. Dựa vào báo cáo của Unit 42, Palo Alto Networks [4] trong số hơn 1,2 triệu thiết bị họ đã nghiên cứu, có đến 98% tất cả các lưu lượng truy cập IoT không được mã hoá, lưu lượng này có thể mang dữ liệu cá nhân và các thông tin nhạy cảm trong mạng. Cũng trong báo cáo này có đến 57% thiết bị IoT dễ bị tấn công ở mức độ nghiêm trọng trung bình hoặc cao, là một môi trường tiềm năng cho các kẻ tấn công.

a) Khái niệm mã độc

Trước hết, thuật ngữ mã độc nói chung trong khoa học máy tính được Cisco [5] định nghĩa như sau: *"Phần mềm độc hại (malware), viết tắt của "malicious software", đề cập đến bất kỳ phần mềm xâm nhập nào được phát triển bởi tội phạm mạng (thường được gọi là "tin tặc") để đánh cắp dữ liệu và làm hỏng hoặc phá hủy máy tính và hệ thống máy tính. Ví dụ về phần mềm độc hại phổ biến bao gồm vi rút, sâu, vi rút Trojan, phần mềm gián điệp, phần mềm quảng cáo và ransomware"*. Một cách tổng quát, mã độc là một chương trình hoặc một đoạn mã được tin tặc bằng một cách bí mật chèn vào các hệ thống máy tính với mục đích độc hại.

Botnet có thể được định nghĩa là một tập hợp các thiết bị bị xâm nhập được gọi là bot chạy mã độc và được kiểm soát bởi quản trị viên được gọi là botmaster. Thông thường, một mạng botnet bao gồm ba thành phần chính: kẻ tấn công; cơ sở hạ tầng độc hại; các bot.



Hình 1.4: Phân tích các mối đe doạ hàng đầu trong IoT [4]

b) Mã độc IoT botnet

Mã độc IoT botnet là loại mã độc được tin tặc phát triển trên nền tảng IoT. TrendMicro [6] đưa ra khái niệm IoT botnet *"Mạng botnet IoT là một mạng lưới các thiết bị được kết nối với Internet vạn vật (IoT), điển hình là các bộ định tuyến, đã bị nhiễm phần mềm độc hại (cụ thể là phần mềm độc hại IoT botnet) và rơi vào tầm kiểm soát của các tác nhân độc hại. Các mạng botnet IoT được biết đến với việc được sử dụng để phát động các cuộc tấn công từ chối dịch vụ (DDoS) phân tán vào các thực thể mục tiêu để làm gián đoạn hoạt động và dịch vụ của họ"*.

Với sự phát triển nhanh và mạnh mẽ nền tảng IoT hiện nay, số lượng thiết bị IoT đang được sử dụng tăng liên tục và vượt qua số lượng máy tính truyền thống rất nhiều. Do đó, ảnh hưởng khi bị IoT botnet tấn công là rất lớn. Trong những năm gần đây, một trong những cuộc tấn công từ chối dịch vụ phân tán (DDoS) lớn nhất sử dụng mạng botnet IoT vào Dyn, công ty cung cấp DNS vào năm 2016 [7]. Chúng tấn công trang web của Dyn với các yêu cầu được xác nhận 1,2 terabit mỗi giây. Mạng botnet IoT trở nên phổ biến do tác động không thể bỏ qua của nó, được gọi là Mirai [8]. Các thiết bị IoT không đồng nhất, việc phát hiện lỗ hổng IoT cũng như IoT Botnet trở nên khó khăn. Vì vậy, phát hiện IoT Botnet là một hướng nghiên cứu rất cần thiết.

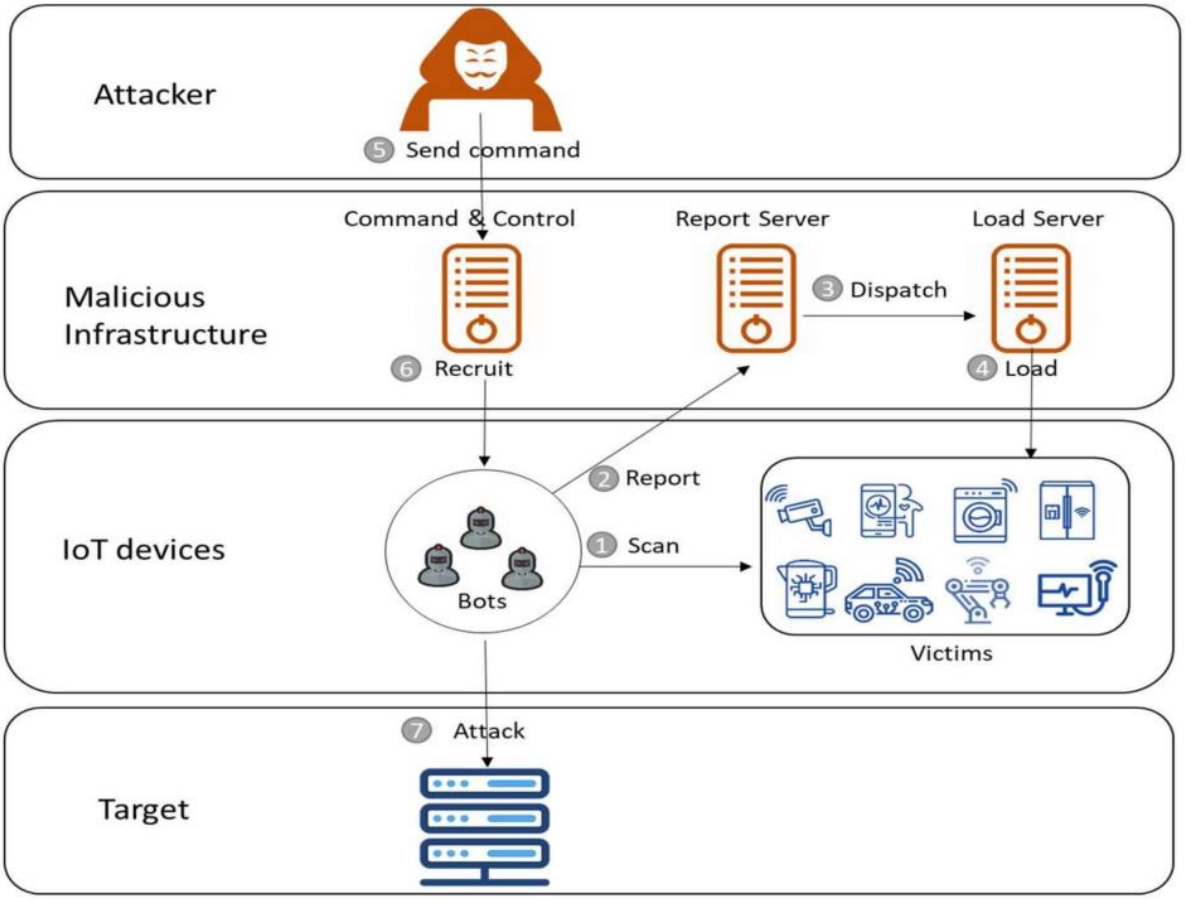
Có nhiều nghiên cứu về IoT botnet [9]–[11] chỉ ra rằng IoT botnet hoạt động qua ba giai đoạn chính như sau:

Giai đoạn 1: Giai đoạn quét: Để xác định vị trí của một thiết bị dễ bị tấn công, một bot (hoặc mã độc hại) thực hiện quét và do thám. Botmaster quét các thiết bị IoT dễ bị tấn công. Một khi nó tìm thấy nó, nó bắt đầu lây nhiễm mã độc thông qua khai thác lỗ hổng. Khi thiết bị bị xâm phạm, nó sẽ trở thành một bot và bắt đầu giao tiếp với botmaster. Thông qua việc sử dụng thông tin đăng nhập yếu hoặc khai thác các lỗ hổng đã biết của các thiết bị IoT, bot sẽ tấn công đến các nạn nhân mới.

Giai đoạn 2: Giai đoạn lan truyền: Một phiên bản phù hợp của bot được cài đặt và thực thi dựa trên kiến trúc của thiết bị dễ bị tấn công. Thông thường, để được quyền kiểm soát hoàn toàn, bot sẽ dừng các quá trình liên kết với dịch vụ liên quan để xóa bất kỳ phần mềm độc hại nào khác phần mềm độc hại trước đó và khóa các cổng vào chính nó. Mã độc lan truyền để mở rộng mạng botnet IoT càng nhanh càng tốt. Trong giai đoạn này, các bot vẫn đang chờ lệnh từ botmaster.

Giai đoạn 3: Giai đoạn tấn công: Thực hiện các hoạt động độc hại như DDoS, khai thác tiền điện tử và thư rác. Kẻ tấn công bắt đầu cuộc tấn công bằng cách gửi các lệnh thông qua máy chủ chỉ huy và điều khiển đến tất cả các bot được phân phối để kích hoạt cuộc tấn công. Do đó, các bot bắt đầu cuộc tấn công sau khi nhận được các lệnh giống hệt nhau.

Hiểu được cách thức hoạt động của các botnet trên Internet là rất quan trọng để tìm ra những cách phát hiện mới, hiệu quả và xử lý chúng nhằm hạn chế thiệt hại. Hiểu hoạt động của các bot này một cách toàn diện hơn sẽ giúp chống lại các cuộc tấn công và giữ cho không gian mạng trong sạch, từ đó giúp đảm bảo an ninh của Internet. Hầu hết các mã độc IoT botnet đều có các bước làm việc tương tự nhau. Hình 1.5 minh họa cách hoạt động của các botnet IoT.



Hình 1.5: Sơ đồ luồng tấn công của một IoT botnet [11]

Các biến chủng mã độc IoT botnet hiện nay ngày càng tiến hoá và tinh vi hơn, đa số được phát triển dựa vào một số cơ sở mã độc mã nguồn mở. Một số cơ sở mã độc được Trend Micro phân tích [12] và là cơ sở để phát triển hầu hết các mã độc IoT botnet hiện nay.

Kaiten (hay còn gọi là Tsunami) được cho là ít được biết đến nhất trong ba cơ sở mật mã. Tuy nhiên, cơ sở mã này, là mã nguồn mở từ năm 2001, vẫn còn phổ biến trong giới tội phạm mạng. Kaiten lây lan bằng các dịch vụ Telnet, các biến thể gần đây của nó có tính năng tiêu diệt bot để làm sạch bất kỳ sự lây nhiễm nào khác trước đó.

Qbot: Mặc dù mới hơn Kaiten, nhưng Qbot cũng là một họ phần mềm độc hại botnet IoT tương đối cũ. Nó xuất hiện lần đầu tiên vào năm 2008, nhưng nó vẫn còn phổ biến trong giới tội phạm mạng. Nó còn được gọi là Bashlite, Gafgyt, Lizkebab,

hoặc Torlus. Giống như Kaiten's, các biến thể của Qbot có tính năng diệt bot giúp gỡ cài đặt các phần mềm độc hại botnet khác.

Mirai: Mirai là cơ sở phổ biến nhất và được biết đến nhiều nhất trong số ba cơ sở mã. Nó nổi lên vào năm 2016, khi nó tạo dựng được tên tuổi sau khi đánh sập các trang web và dịch vụ lớn. Nó được thiết kế như một công cụ DDoS để bán và được sử dụng để nhắm mục tiêu đến các game thủ. Một số biến thể của Mirai có khả năng dọn dẹp các phần mềm lây nhiễm cũ hơn và hoàn toàn độc quyền một thiết bị.

1.2 Tổng quan hệ thống phân tích và phát hiện mã độc IoT Botnet

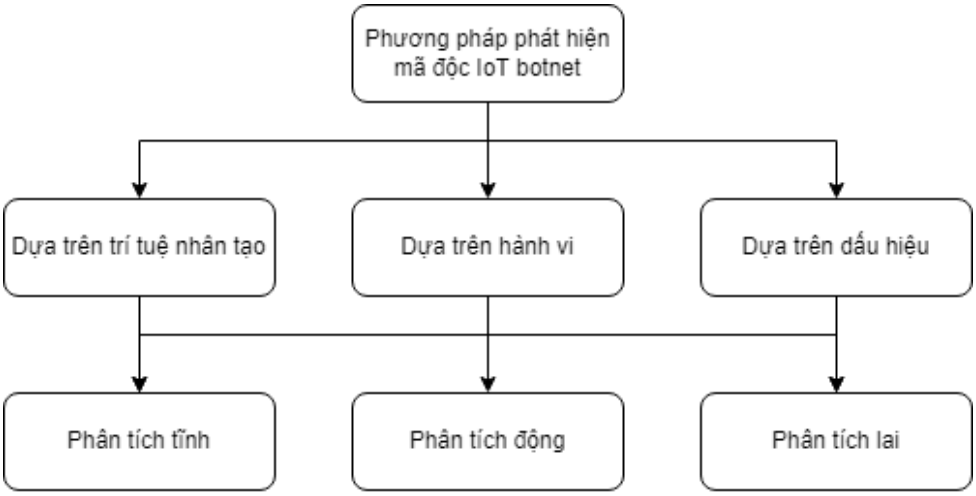
1.2.1 Các phương pháp phân tích và phát hiện mã độc IoT Botnet

Hiện nay, các phương pháp phát hiện mã độc được nghiên cứu và áp dụng trên thế giới được chia thành ba loại chính như sau: (1) các phương pháp phát hiện dựa trên trí tuệ nhân tạo (AI-based), (2) các phương pháp phát hiện dựa trên hành vi (behavior-based) và (3) các phương pháp phát hiện dựa trên dấu hiệu (signature-based).

Khái niệm đằng sau trí tuệ nhân tạo và các nhánh của nó (học máy và học sâu), cung cấp một thuật toán có thể phân tích thông tin và nhận dạng các mẫu, từ đó xây dựng một mô hình mà máy có thể sử dụng để phân tích thông tin mà nó chưa từng thấy trước đây. Thuật toán học liên tục và có thể đưa ra các quyết định đáng tin cậy lặp đi lặp lại khi các hệ thống cung cấp nhiều dữ liệu huấn luyện cho mô hình. Phương pháp phát hiện dựa trên trí tuệ nhân tạo sử dụng các đặc trưng của mã độc từ đó đưa ra một mô hình dự đoán và phát hiện sớm mã độc. Các mô hình được sử dụng trong phát hiện IoT botnet dựa trên các thuật toán học máy như học có giám sát, học không giám sát, học sâu.

Các phương pháp (2) không sử dụng thuật toán trí tuệ nhân tạo và thay vào đó sử dụng các phương pháp dựa trên đặc điểm kỹ thuật để phát hiện. Nói chung, một phương pháp phát hiện đặc điểm kỹ thuật phụ thuộc vào thông số kỹ thuật mô tả hành vi dự kiến. Nó có thể phát hiện các cuộc tấn công chưa từng gặp phải trước đó và có tỷ lệ dương tính giả thấp, nhưng nó có nhược điểm là kém hiệu quả hơn các phương pháp phát hiện bất thường và tốn nhiều thời gian hơn.

Các phương pháp dựa trên dấu hiệu (3) giữ lại cơ sở dữ liệu của các kỹ thuật xâm nhập đã biết (chữ ký tấn công) và phát hiện sự xâm nhập bằng cách so sánh các hành vi với cơ sở dữ liệu. Chúng phát hiện chính xác các cuộc tấn công đã biết và yêu cầu ít tài nguyên hơn để phát hiện các cuộc xâm nhập. Chúng có nhược điểm là không hiệu quả trong việc phát hiện các cuộc tấn công không xác định hoặc biến thể mới của mã độc.



Hình 1.6: Các phương pháp phát hiện mã độc IoT botnet

Quá trình mổ xẻ mã độc để hiểu cách thức hoạt động, xác định chức năng, nguồn gốc và tác động tiềm ẩn của nó được gọi là phân tích mã độc. Với hàng triệu mã độc mới và các phiên bản biến đổi của các mã độc đã phát hiện trước đó, tổng số mã độc mà các nhà phân tích bảo mật gặp phải đã tăng lên rất lớn trong những năm qua. Có hai cách tiếp cận cơ bản để phân tích mã độc IoT botnet: (1) phân tích tĩnh, (2) phân tích động và (3) phân tích lai. Phân tích tĩnh liên quan đến việc kiểm tra phần mềm độc hại mà không cần chạy nó. Mặt khác, phân tích động liên quan đến việc chạy các phần mềm độc hại có các đặc điểm hệ thống như thế nào và phân tích lai là sự kết hợp giữa phân tích tĩnh với phân tích động.

Phân tích tĩnh bao gồm việc kiểm tra mã hoặc cấu trúc của tệp thực thi mà không thực thi mã độc. Loại phân tích này có thể xác nhận xem một tệp có độc hại hay không, cung cấp thông tin về chức năng và cũng có thể được sử dụng để tạo ra một bộ chữ ký đơn giản.

Phân tích động liên quan đến việc thực thi chương trình và giám sát hành vi của nó trên hệ thống. Không giống như phân tích tĩnh, nó theo dõi các hành động thực được thực hiện bởi chương trình.

Cách tiếp cận phân tích lai kết hợp giữa phân tích tĩnh và phân tích động. Trong khi mỗi cách tiếp cần đều có ưu điểm và hạn chế nhất định, cách tiếp cận phân tích lai có thể kết hợp các ưu điểm của cả phân tích tĩnh và phân tích động. Khi đó có thể nâng cao hiệu quả của việc phát hiện mã độc IoT botnet. Các đặc trưng của phân tích tĩnh và phân tích động được sử dụng, sau đó kết hợp lại bằng một thuật toán hoặc mô hình nhất định để ra đặc trưng sau khi phân tích lai. Tuy nhiên phân tích lai cũng có nhược điểm, do kết hợp giữa các cách tiếp cận nên phân tích lai phức tạp hơn so với các cách phân tích còn lại. Ngoài ra việc kết hợp này cũng cần không gian lưu trữ và tài nguyên tính toán lớn.

Bảng 1.1 So sánh giữa các phương pháp phân tích mã độc IoT botnet

	Phân tích tĩnh	**Phân tích động**	**Phân tích lai**
Ưu điểm	- Phân tích chi tiết các tập tin, đoạn mã và đưa ra cấu trúc mã độc. - Không cần thực thi mã độc.	- Thực thi chương trình để theo dõi và quyết định tập tin có phải mã độc không.	- Kết hợp ưu điểm về đặc trưng của hai phương pháp. - Kết quả phân tích tổng quát về mã độc hơn.
Hạn chế	- Cần dịch ngược mã máy nên cần các công nghệ, phần mềm hỗ trợ dịch ngược.	- Có nguy cơ mất an toàn hệ thống do phải thực thi mã độc. - Khó mô phỏng đầy đủ các kiến trúc phần cứng IoT.	- Kiến trúc xử lý phức tạp hơn. - Cần nhiều không gian lưu trữ và tài nguyên tính toán.

1.2.2 Học máy trong phát hiện mã độc IoT Botnet

Trí tuệ nhân tạo đã và đang được nghiên cứu ứng dụng rộng rãi trong các lĩnh vực, trong đó có bảo mật trong IoT. Các phương pháp học máy được áp dụng để phát hiện các cuộc tấn công sớm dựa trên các dữ liệu đặc trưng của hệ thống hoặc tập tin. Mã độc IoT hiện nay có rất nhiều biến chủng tinh vi và số lượng lớn, vì vậy các mô hình học máy cũng cần được cập nhật thường xuyên. Các thuật toán học máy được chia làm 3 loại, gồm:

- Học có giám sát (Supervised learning) là nhiệm vụ học máy học một hàm chức năng ánh xạ đầu vào đến đầu ra dựa trên các cặp đầu vào - đầu ra huấn luyện. Mô hình học máy đưa ra một hàm chức năng từ dữ liệu đào tạo được gắn nhãn. Học có giám sát thường được sử dụng trong các bài toán phân lớp, hồi quy.

- Học không giám sát (Unsupervised learning) là trong quá trình học, thuật toán học các mẫu từ dữ liệu không được gắn nhãn. Quá trình học xây dựng mô hình tìm kiếm cấu trúc hoặc phân ra các cụm có đặc trưng tương đối giống nhau. Học không giám sát thường được sử dụng trong các bài toán phân cụm.

- Học tăng cường (RL) là một loại kỹ thuật học máy cho phép các tác tử (agent) học trong môi trường tương tác bằng cách thử và sai bằng cách sử dụng phản hồi từ các hành động và kinh nghiệm của chính nó.

Một số nghiên cứu ứng dụng học máy trong phát hiện mã độc IoT botnet như tác giả Nguyen và cộng sự [13] đã đề xuất một nỗ lực sử dụng phương pháp phân tích lai. Cơ chế tích phân, quy trình tiền xử lý và phân loại dựa trên các tính năng tương tác có đồ thị thông tin chuỗi có thể in được (đồ thị PSI) và đồ thị gọi hệ thống (SCG). Bộ phân loại được sử dụng thuật toán học máy có giám sát bao gồm Cây quyết định (DT), k-Nearest Neighbor (KNN), SVM (hạt nhân RBF) và Rừng ngẫu nhiên (RF).

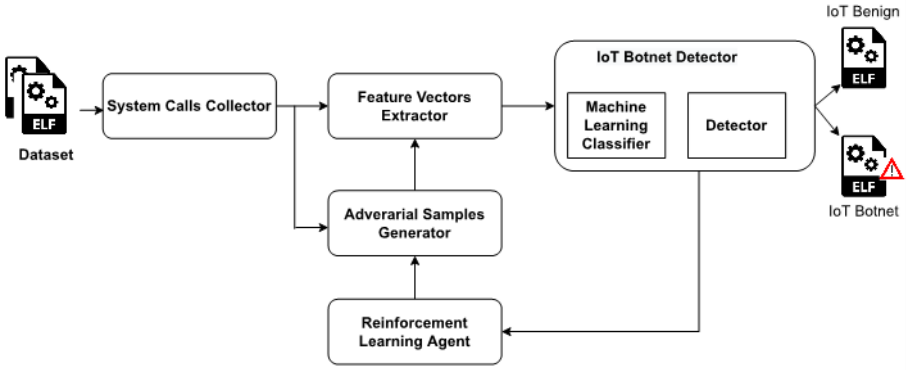
Với học không giám sát, có một số công trình như của tác giả Ozawa và cộng sự trong bài báo [14], các tác giả đã giới thiệu một phương pháp sử dụng học quy tắc kết hợp để tìm ra từ dữ liệu được thu thập trên quy mô lớn từ các darknet, với một luồng lớn, tính đồng nhất của các cuộc tấn công. Họ có thể phát hiện ra các hành vi tấn công máy chủ liên quan đến các nhóm phần mềm độc hại được công nhận bằng cách phát hiện ra sự đối xứng trong các dấu hiệu liên quan đến IoT, chẳng hạn như cổng đích, loại hoạt động và kích thước cửa sổ TCP.

Cuối cùng, với học tăng cường cũng được ứng dụng để tăng cường mô hình học máy phát hiện IoT botnet như công trình của tác giả Ngo và cộng sự [13] đã đề xuất một cách tiếp cận sử dụng mô hình học tăng cường để tạo ra các mẫu đối nghịch sau đó huấn luyện bộ phân loại học máy đồ thị PSI. Mô hình này được đánh giá bằng tập dữ liệu với 10010 mẫu đồ thị PSI bao gồm 6165 mẫu botnet IoT và 3845 mẫu

lành tính. Bộ phân loại đạt độ chính xác 95,13% và từ 40% đến 94,10% sau khi được huấn luyện bằng cách thêm hơn 4083 mẫu đối nghịch vào tập dữ liệu gốc.

1.2.3 Mô hình hệ thống phân tích và phát hiện mã độc IoT Botnet

Ở khuôn khổ luận văn này, học viên trình bày mô hình hệ thống phân tích và phát hiện mã độc do học viên và nhóm của T.S Ngô Quốc Dũng đã nghiên cứu bao gồm chi tiết về giai đoạn tiền xử lý dữ liệu, cơ chế phân loại dựa trên vectơ tính năng và mô hình học tập củng cố để cải thiện phát hiện tấn công zero-day. Mô hình tổng quan hệ thống được minh hoạ tại hình 1.7.



Hình 1.7. Mô hình hệ thống phân tích và phát hiện mã độc IoT botnet

Mô hình được đề xuất bao gồm năm thành phần chính: Bộ thu thập lệnh gọi hệ thống (SCC – System Calls Collector), Bộ trích xuất vectơ tính năng (FVE – Feature Vectors Extractor), Bộ phát hiện Botnet IoT (IBD – IoT Botnet Dectector), Bộ tạo mẫu đối nghịch (ASG – Adversarial Samples Generator), Tác nhân học tăng cường (RLA – Reinforcement learning). Hoạt động của mô hình được chia thành hai giai đoạn:

(1) Giai đoạn đầu tiên là giai đoạn phân loại học máy mà IBD sẽ được đào tạo bằng cách sử dụng các bộ phân loại học máy khác nhau. Thứ nhất, phân tích động được sử dụng để xử lý tập dữ liệu. Các tệp Định dạng liên kết mở rộng (ELF - Extensible Linking Format) được SCC xử lý bằng cách sử dụng môi trường sandbox, từ đó, đồ thị lệnh gọi hệ thống là kết quả đầu ra của sandbox. Các lệnh gọi hệ thống đó được FVE phân tích bằng thuật toán graph2vec [15, p. 2]. Sau đó, dữ liệu đồ họa của SCG được biến đổi thành vector đặc trưng với kích thước cố định. Bước cuối

cùng của giai đoạn này, IBD sử dụng một số thuật toán học máy để phân loại dữ liệu này là lành tính hay phần mềm độc hại.

(2) Giai đoạn thứ hai là cải thiện phân loại, IBD được phát triển bằng cách học tăng cường để phát hiện cuộc tấn công zero-day. Ban đầu, RLA sử dụng phương pháp học tăng cường để tìm ra cách tốt nhất để tạo một cấu trúc đồ thị gọi hệ thống mới. Sau đó, ASG kết hợp đồ thị cuộc gọi hệ thống phần mềm độc hại ban đầu với đồ thị lệnh gọi hệ thống mới từ RLA. Để đảm bảo các thuộc tính độc hại của mã độc, ASG thêm đồ thị lệnh gọi mới từ RLA vào nút đầu tiên của đồ thị lệnh gọi mã độc để tạo một mẫu SCG chưa được huấn luyện. Tiếp theo, mẫu SCG mới chưa được huấn luyện được phân tích và trích xuất các vectơ đặc trưng giống như giai đoạn đầu. Sau đó, IBD thực hiện phân loại các mẫu SCG mới được tạo ra. Kết quả của IBD là phần thưởng trả về RLA, kết quả có thể là dương nếu mẫu đối nghịch này gây ra sự phân loại sai từ IBD và ngược lại. Cuối cùng là đánh giá mô hình bằng cách sử dụng bộ dữ liệu thử nghiệm với các mẫu zero-day để đánh giá hiệu quả và độ chính xác. Chi tiết quá trình xử lý của mô hình như sau:

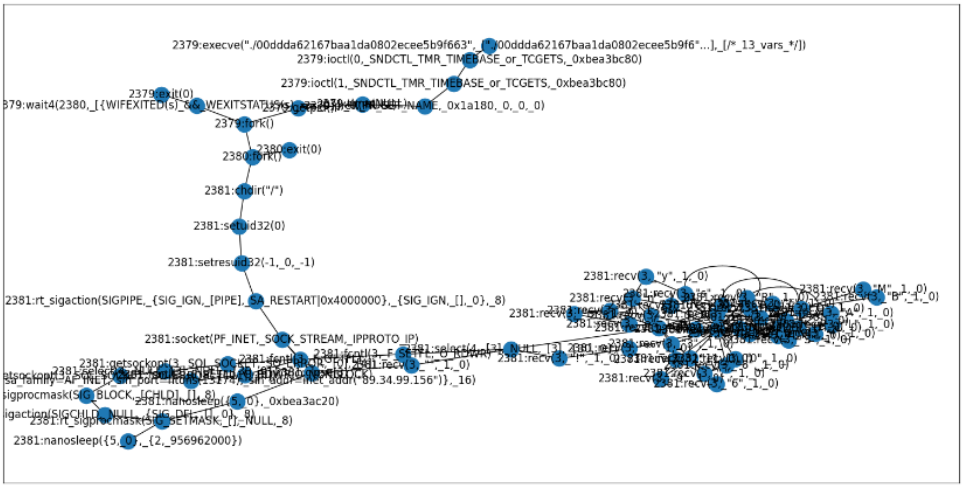
a) Bộ thu thập lệnh gọi hệ thống (SCC – System Calls Collector):

Phương pháp phân tích động được sử dụng để theo dõi hành vi độc hại của IoT Botnet và thu thập nhật ký cuộc gọi hệ thống. Ban đầu, các tệp ELF được thực thi trong môi trường sandbox có các thuộc tính tương tự với thiết bị IoT. Và đầu ra là bản ghi dữ liệu cuộc gọi hệ thống mang theo thông tin cuộc gọi của hệ thống. Lệnh gọi hệ thống này ghi lại sự tương tác của mạng botnet IoT với các hành vi độc hại như cố gắng quét cổng, tấn công brute-force, cố gắng kết nối với máy chủ C&C, phát hiện các thiết bị khác thông qua các cổng dịch vụ mở khác nhau. Để đảm bảo điều kiện tốt nhất của IoT Botnet, hộp cát yêu cầu khả năng hỗ trợ đa kiến trúc, cung cấp thư viện động, trình giả lập yêu cầu máy chủ C&C. Do đó, học viên đã sử dụng V-Sandbox [16] để thu thập thông tin về hành vi độc hại. Sau đó, các tệp nhật ký cuộc gọi hệ thống được xử lý để tạo SCG. Trong khuôn khổ luận văn, một SCG được định nghĩa như sau:

Định nghĩa 2: SCG là một đồ thị có hướng được định nghĩa là $G_{SCG}(V, E)$ *trong đó:*

- *V là các đỉnh được thiết lập bởi các lệnh gọi hệ thống với id tiến trình, tên và đối số.*
- *E là tập các cạnh mà mỗi cạnh nối từ đỉnh* v_i *đến đỉnh* v_j *của đồ thị phản ánh mối quan hệ của hai lệnh gọi hệ thống. Với khuyên, nó chỉ được tính là một cạnh.*

Bước tiếp theo, SCG được trích xuất thông qua dữ liệu cuộc gọi hệ thống bằng cách xem thứ tự cuộc gọi của mẫu đầu vào từ V-Sandbox. SCG bao gồm các đỉnh được gắn nhãn id quy trình (PID), tên, đối số và các cạnh là kết nối giữa các đỉnh. Nhánh đồ thị với quy trình con này hoàn toàn có thể phân tích để phát hiện hành vi của IoT Botnet. Hình 1.8 minh họa phần mềm độc hại SCG bắt đầu từ lệnh gọi hệ thống execve().



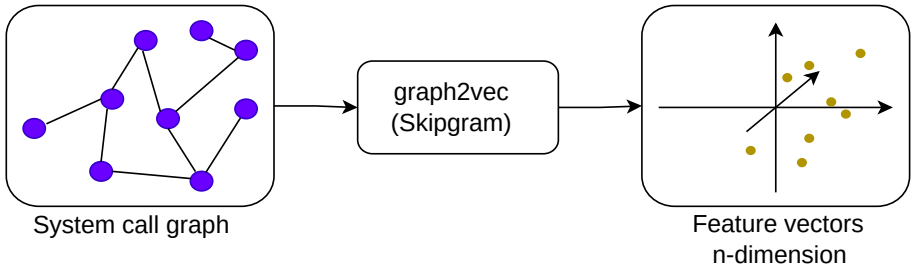
Hình 1.8. Đồ thị các lệnh gọi hệ thống của một tệp tin ELF chứa mã độc.

b) Trích xuất các vectơ đặc trưng của đồ thị lệnh gọi hệ thống.

Để thích ứng với đầu vào của bộ phân loại học máy, dữ liệu SCG thể hiện dưới dạng vectơ đặc trưng với độ dài cố định với mục đích trích xuất đặc trưng dữ liệu cũng như giảm thiểu thời gian và chi phí tính toán. Tại đây, học viên đã sử dụng kỹ thuật nhúng đồ thị (graph embedding) để giải quyết vấn đề mỗi SCG mang thuộc tính của mỗi tệp ELF đầu vào với số đỉnh và số cạnh khác nhau. Toàn bộ đồ thị được

chuyển đổi thành một vectơ đặc trưng, sau đó nó được cung cấp cho các bộ phân loại học máy. Trong khuôn khổ bài báo, học viên sử dụng framework Graph2vec [15] do Narayanan và cộng sự đề xuất. Tác giả coi toàn bộ đồ thị là một tài liệu và các đồ thị con gốc xung quanh mọi nút trong đồ thị là một từ, xử lý đồ thị bằng cách mở rộng phương pháp Skipgram cho văn bản (Doc2vec). Việc nhúng đồ thị được học theo phương pháp học không giám sát.

Trong giai đoạn đầu tiên, các đồ thị lệnh gọi hệ thống ban đầu được tạo bởi SCC được trích xuất thành vector đặc trưng và sau đó trở thành đầu vào của bộ phân loại. Mặt khác, trong giai đoạn thứ hai, FVE sẽ trích xuất các vectơ đặc trưng từ các mẫu đối nghịch có chứa đồ thị lệnh gọi hệ thống đã sửa đổi và đưa chúng vào bộ phân loại một lần nữa. Việc trích xuất tính năng được minh họa như Hình 1.9.



Hình 1.9. Trích xuất vector đặc trưng của đồ thị lệnh gọi hệ thống

c) Học tăng cường

Tác tử học tăng cường được sử dụng trong giai đoạn thứ hai, sẽ cải thiện trình phân loại bằng cách sử dụng mẫu đối nghịch, có chức năng tạo đồ thị lệnh gọi hệ thống mới. Sau đó cung cấp đồ thị lệnh gọi hệ thống mới cho ASG. Đồ thị lệnh gọi hệ thống ban đầu được tạo với tất cả các đỉnh là lệnh gọi hệ thống lành tính. Sau đó, RLA cố gắng sửa đổi đồ thị lệnh gọi hệ thống này bằng cách thêm các đỉnh và cạnh giả để tạo ra một đồ thị lệnh gọi hệ thống lành tính mới. Để tìm ra phương pháp tạo đồ thị tối ưu, mô hình học tăng cường được áp dụng là Q-learning. Mô hình học tăng cường xác định như sau:

- Môi trường (Environment) có trách nhiệm trả lại phần thưởng cho tác tử. Sự tương tác của môi trường với tác tử bằng cách đưa ra phần thưởng lớn

khi tác tử thực hiện một số hành động khiến IBD phân loại kết quả sai và nếu không sẽ trừng phạt tác tử bằng phần thưởng âm.

- Hành động (Action) là phương pháp tác tử chọn để sửa đổi cấu trúc của đồ thị bằng cách thêm một số cạnh vào đồ thị. Tập hợp hành động là A = $\{a_1, a_2, a_3, \dots, a_n\}$ tương ứng thêm một số cạnh vào đỉnh a.
- Trạng thái (State) là tập các đỉnh của đồ thị lệnh gọi hệ thống ban đầu. Tập trạng thái là $S = \{x_0, x_1, \dots, x_N\}$ trong đó N là số đỉnh. Tác tử chuyển sang trạng thái mới sau khi thực hiện một hành động. Trạng thái được coi là trạng thái kết thúc khi lệnh gọi hệ thống mới có thể đánh lừa bộ phân loại hoặc vượt quá số bước nhất định.
- Phần thưởng (Reward) *r* là 1 khoản thưởng mà tác tử nhận được khi thực hiện một hành động. Nếu IBD phân loại kết quả sai với hành động của tác tử, thì đó là hành động đúng và tác tử sẽ nhận được phần thưởng dương (*r = 1000*). Ngược lại, với mỗi vòng lặp và IBD phân loại kết quả đúng, hình phạt sẽ được đặt ra với *r = -1*.

Mục đích của mô hình học tăng cường là làm cho IBD phân loại sai bằng cách tạo ra đồ thị gọi hệ thống lành tính giả mới. Đồ thị cuộc gọi hệ thống này được kết hợp với đồ thị cuộc gọi hệ thống mã độc ban đầu trong giai đoạn đầu tiên và sau đó trích xuất thành vectơ đặc trưng, cuối cùng được cung cấp cho bộ phân loại.

d) Phân lớp sử dụng các thuật toán học máy.

Sau khi tập dữ liệu được FVE xử lý trước, đầu vào của bộ phân loại là các vectơ đặc trưng có độ dài cố định. Cuối cùng, tập dữ liệu chuẩn hóa được cung cấp cho bộ phân loại học máy để huấn luyện và thử nghiệm. Trong bài báo này sử dụng các nhịp thuật toán học máy có giám sát phổ biến như Naive Bayes (NB), SVM, KNN, Cây quyết định (Decision Tree), Rừng ngẫu nhiên (Random Forest).

Bộ phân loại đã được huấn luyện theo hai giai đoạn để cải thiện khả năng phát hiện cuộc tấn công zero-day. Thứ nhất, tập dữ liệu mặc định được cung cấp cho các bộ phân loại, các mẫu IoT Botnet đã biết của tập dữ liệu mặc định sau đó có thể được phát hiện bởi các bộ phân loại. Cuối cùng, tập dữ liệu mới được RLA và ASG sửa

đổi nhưng vẫn giữ nguyên hành vi độc hại, được cung cấp cho bộ phân loại. Tập dữ liệu đối nghịch mới có thể cải thiện khả năng phát hiện tấn công zero-day.

1.3. Tổng quan giải pháp xử lý phân tán động

1.3.1. Các định nghĩa về hệ thống phân tán.

Với việc xử lý một lượng lớn dữ liệu và cần tài nguyên tính toán lớn, việc thiết kế và xây dựng một hệ thống cân bằng năng lực giữa các thiết bị IoT Analyzer dựa trên nền tảng hệ thống phân tán là rất cần thiết. Hệ thống phân tán hiện nay được định nghĩa theo nhiều cách, trong đó có một số định nghĩa tương đối tổng quát như của tác giả George Coulouris và cộng sự [17] đã định nghĩa: *"hệ thống phân tán là một hệ thống trong đó các thành phần phần cứng hoặc phần mềm được kết nối mạng được giao tiếp và điều phối tác vụ qua cách truyền các message."*. Tác giả Andrew S. Tanenbaum và cộng sự [18] đã đưa ra định nghĩa về hệ thống phân tán như sau: *"Một hệ thống phân tán là một tập hợp các máy tính độc lập nhưng ở phía người dùng của nhìn nhận như một hệ thống mạch lạc duy nhất."*. Qua các định nghĩa khác nhau về hệ thống phân tán, trong khuôn khổ luận văn, học viên định nghĩa hệ thống phân tán như sau:

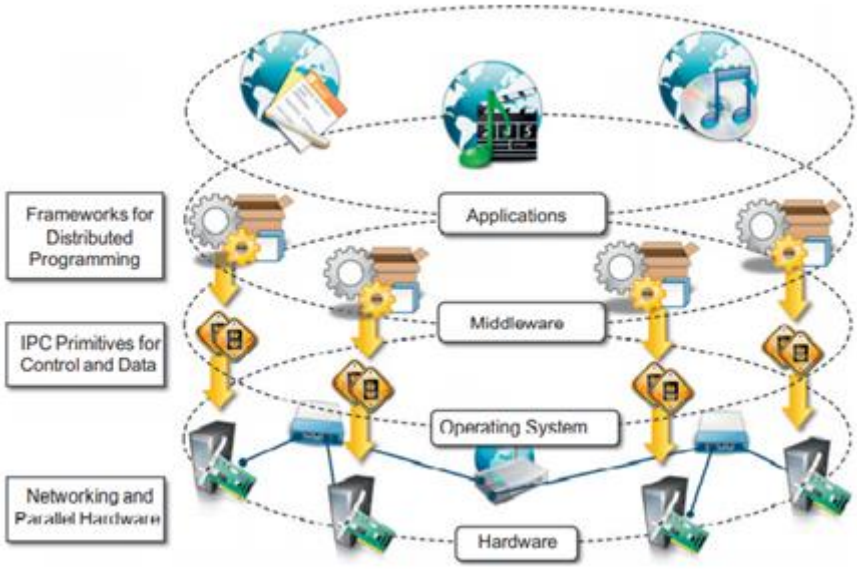
Định nghĩa 3: Hệ thống phân tán là hệ thống mà các máy tính được kết nối với nhau, trong đó các tác vụ được điều phối xử lý phân tán giữa các máy và ở phía người dùng thì nhìn nhận như một hệ thống đồng nhất.

Các hệ thống tính toán song song chỉ là một trường hợp cụ thể của tính toán phân tán. Trong tính toán song song, sự song song đạt được thông qua việc chia chỏ một khối lệnh lớn thành nhiều lệnh con và thực thi các lệnh con này đồng thời trên nhiều nhân xử lý của CPU

1.3.2 Các thành phần trong một hệ thống phân tán

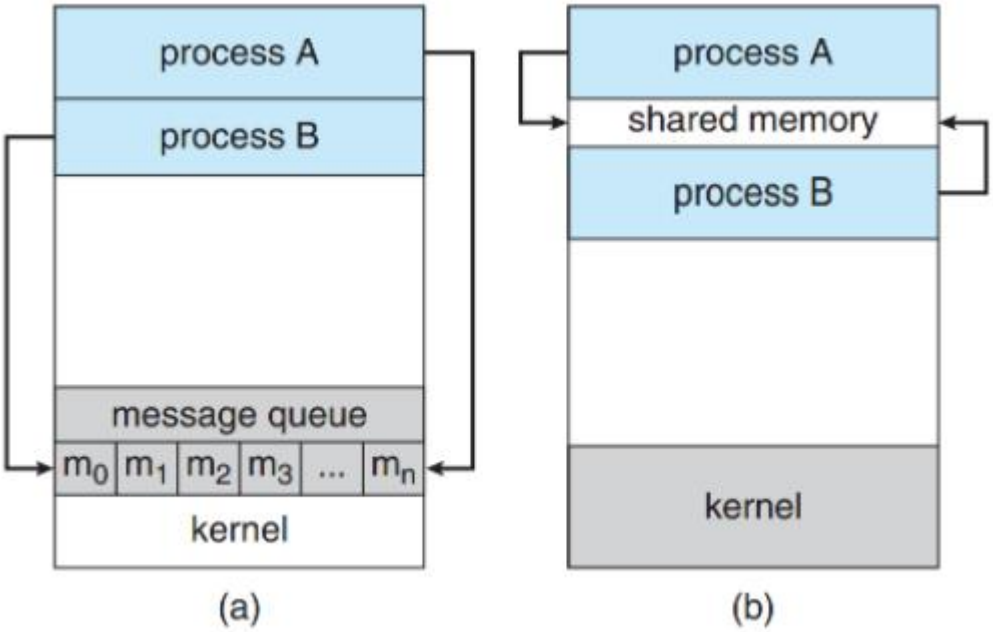
Trong hệ thống phân tán, việc chia sẻ bộ nhớ và cơ chế liên lạc giữa các thành phần trong hệ thống là hai yếu tố quan trọng nhất. Căn cứ theo một số định nghĩa, khái niệm và các đặc điểm của hệ thống phân tán thì cơ chế liên lạc giữa các cấu phần (cả phần cứng và phần mềm) trong hệ thống phân tán là yếu tố quan trọng nhất. Nếu coi mức trừu tượng thấp nhất trong hệ thống phần tán là mức tiến trình (process) thì

cơ chế liên lạc giữa các cấu phần của hệ thống phân tán là IPC – Inter Process Communication Hình 1.10 minh hoạ mô hình của một hệ thống phân tán.



Hình 1.10 Mô hình phân lớp của hệ thống phân tán.

Cơ chế liên lạc giữa các thành phần trong hệ thống phân tán tại lớp trung gian (middle ware) được triển khai cơ chế Inter Process Communuication (IPC). Tại đây, lớp trừu tượng thấp nhất của hệ thống phân tán được coi là mức tiến trình (process). IPC được coi là trung tâm của tất cả các hệ thống phân tán và vì vai trò nổi bật của nó, rất nhiều tài liệu nghiên cứu đã từng được thực hiện dựa trên các phương pháp và cơ chế của IPC. Hai mô hình chính thuộc cơ chế liên lạc IPC là mô hình chia sẻ bộ nhớ (shared memory) – với nhiệm vụ định nghĩa, cấp phát và khởi tạo khu vực bộ nhớ lưu trữ chung – và mô hình truyền nhận thông điệp (message passing) – với nhiệm vụ truyền tải thông điệp giữa các tiến trình. Cả hai mô hình IPC trên đều được sử dụng phổ biến trong các hệ điều hành. Mô hình truyền nhận thông điệp (message passing) tỏ ra hữu ích cho việc trao đổi dữ liệu dung lượng nhỏ và dễ thực hiện hơn trong hệ thống phân tán (distributed system). Ngược lại, mô hình chia sẻ bộ nhớ (shared memory) có thể đạt tốc độ xử lý nhanh hơn mô hình truyền nhận dữ liệu (message passing) vì các hệ thống truyền thông điệp thường thực hiện thông qua system call (tốn nhiều thời gian hơn và phải có sự can thiệp của kernel – nhân hệ điều hành).



Hình 1.11 Hai mô hình cơ bản của IPC

a) Mô hình chia sẻ bộ nhớ (shared-memory system)

Các tiến trình được đặt trong các không gian địa chỉ khác nhau để không ảnh hưởng tới nhau. Hạn chế chính của sự cô lập này là nếu một tiến trình yêu cầu truyền một số dữ liệu cho một tiến trình khác, dữ liệu phải được sao chép, đây có thể là một hoạt động tương đối tốn thời gian cho một lượng dữ liệu to lớn. Để xử lý vấn đề đó, bộ nhớ chia sẻ được sử dụng. Như tên của nó, thông qua bộ nhớ chia sẻ, hai hoặc nhiều tiến trình có quyền truy cập vào cùng một vị trí bộ nhớ và có thể truyền dữ liệu. Bộ nhớ chia sẻ không xử lý các vấn đề đồng thời cho các tiến trình liên quan. Để đạt được mục tiêu này, nó thường khai thác các kỹ thuật kiểm soát đồng thời như Semaphore. Một trong những lợi ích đáng kể của chia sẻ bộ nhớ là khi các tiến trình trao đổi một lượng lớn dữ liệu.

Có hai loại chia sẻ bộ nhớ sẽ được nghiên cứu như sau:

- Ánh xạ tệp tin (Mapped file): vùng của bộ nhớ ảo thuộc về tiến trình được ánh xạ vào các tệp. Theo cách nói khác, đọc hoặc ghi vào các phần bộ nhớ đó được ánh xạ tới đọc hoặc ghi các hoạt động vào tệp. Cách tiếp cận này là loại ánh xạ mặc định. Có hai loại ánh xạ tệp tin như sau: (1) Persisted: Trong loại này, khi quá trình cuối cùng bị chấm dứt, dữ liệu được lưu vào tệp nguồn trên đĩa. Các tệp ánh xạ

bộ nhớ này phù hợp để quản lý các tệp nguồn lớn. (2) Non-Persisted: Trong loại này, khi tiến trình cuối cùng làm việc với một tệp được kết thúc, dữ liệu sẽ bị xóa. Các tệp ánh xạ bộ nhớ này thuận tiện khi bộ nhớ được chia sẻ được sử dụng để giao tiếp giữa các tiến trình.

- Ánh xạ vô danh (Anonymous Mapping): Trong loại ánh xạ này, khu vực của bộ nhớ ảo thuộc sở hữu của một tiến trình được ánh xạ. Các nội dung được đặt thành 0. Ánh xạ này giống như phân bổ bộ nhớ động. Bộ nhớ trong một ánh xạ tiến trình có thể được chia sẻ với các ánh xạ liên quan đến các tiến trình khác. Điều này có thể được thực hiện thông qua hai cách tiếp cận: (1) Nếu một phân đoạn của một tệp được ánh xạ bởi hai tiến trình, cùng một trang bộ nhớ vật lý được chia sẻ bởi chúng. (2) Nếu một tiến trình con được xây dựng, nó kế thừa các ánh xạ thuộc về cha mẹ của nó liên kết với cùng một trang của bộ nhớ vật lý của cha mẹ đó. Khi bất kỳ sửa đổi nào được thực hiện trên dữ liệu trong tiến trình con, các trang khác nhau sẽ được thực hiện tiến trình con. Khi hai hoặc nhiều tiến trình chia sẻ cùng một trang, mỗi tiến trình có thể phát hiện các thay đổi trong nội dung trang được thực hiện bởi các tiến trình khác tùy thuộc vào loại ánh xạ.

b) Mô hình truyền thông điệp (massage passing)

Bên cạnh việc dùng mô hình chia sẻ bộ nhớ - shared memory, một cách khác để liên kết các tiến trình lại với nhau là sử dụng cơ chế truyền nhận thông điệp - message passing. Trong IPC, mô hình truyền thông điệp cũng được nghiên cứu để liên kết các tiến trình với nhau. Trong mô hình truyền thông điệp, các tiến trình giao tiếp bằng cách truyền tin nhắn chỉ bằng hai thao tác: Gửi và nhận. Khái niệm truyền thông điệp có vẻ tương đối đơn giản, nhưng nó đòi hỏi nhiều tùy chọn thiết kế phải được thực hiện. Có nhiều phương pháp truyền thông điệp trong đó có 2 cách thức là kết nối trực tiếp và kết nối gián tiếp.

Cơ chế truyền nhận thông điệp (message passing) cung cấp một phương pháp cho phép các tiến trình liên lạc và đồng bộ mà không cần chia sẻ không gian bộ nhớ của nhau. Điều này đặc biệt hữu ích trong những hệ cơ sở dữ liệu phân tán (distributed

database), nơi mà các tiến trình nằm trên các máy tính khác nhau kết nối qua hệ thống mạng.

Tiến trình gửi tin có thể cố định hoặc biến đổi về kích thước. Ví dụ, nếu 2 tiến trình P và Q muốn trao đổi, các tin cần phải được gửi và nhận giữa hai đầu: một liên kết truyền tin phải tồn tại giữa hai process.

Các hệ thống hoạt động dựa trên truyền nhận thông điệp gồm 2 cách thức: kết nối trức tiếp và kết nối gián tiếp. Do đó, đây là tầng đóng vai trò quan trọng nhất trong việc thể hiện tính phân tán của hệ thống dưới góc độ phần mềm

1.3.3. Tổng quan về lý thuyết cân bằng tải.

Load balancing (cân bằng tải) là một phương pháp phân phối khối lượng tải trên nhiều máy tính hoặc một cụm máy tính để có thể sử dụng tối ưu các nguồn lực, tối đa hóa thông lượng, giảm thời gian đáp ứng và tránh tình trạng quá tải trên máy chủ. Lợi ích của cân bằng tải giúp cân bằng tải làm tăng khả năng đáp ứng, tránh tình trạng quá tải của hệ thống máy tính, đồng thời duy trì tính linh hoạt và khả năng mở rộng cho hệ thống. So với những hệ thống không sử dụng cân bằng tải, hệ thống sử dụng cân bằng tải tỏ ra ưu việt hơn hẳn về nhiều mặt như: trải nghiệm khách hàng, tốc độ xử lý, tính sẵn sàng của dịch vụ… Cụ thể như sau:

Tăng độ tin cậy và khả năng dự phòng cho hệ thống: sử dụng cân bằng tải giúp tăng tính sẵn sàng (availability) cho hệ thống, đồng thời đảm bảo cho người dùng không bị gián đoạn dịch vụ khi xảy ra lỗi sự cố lỗi tại một điểm cung cấp dịch vụ.

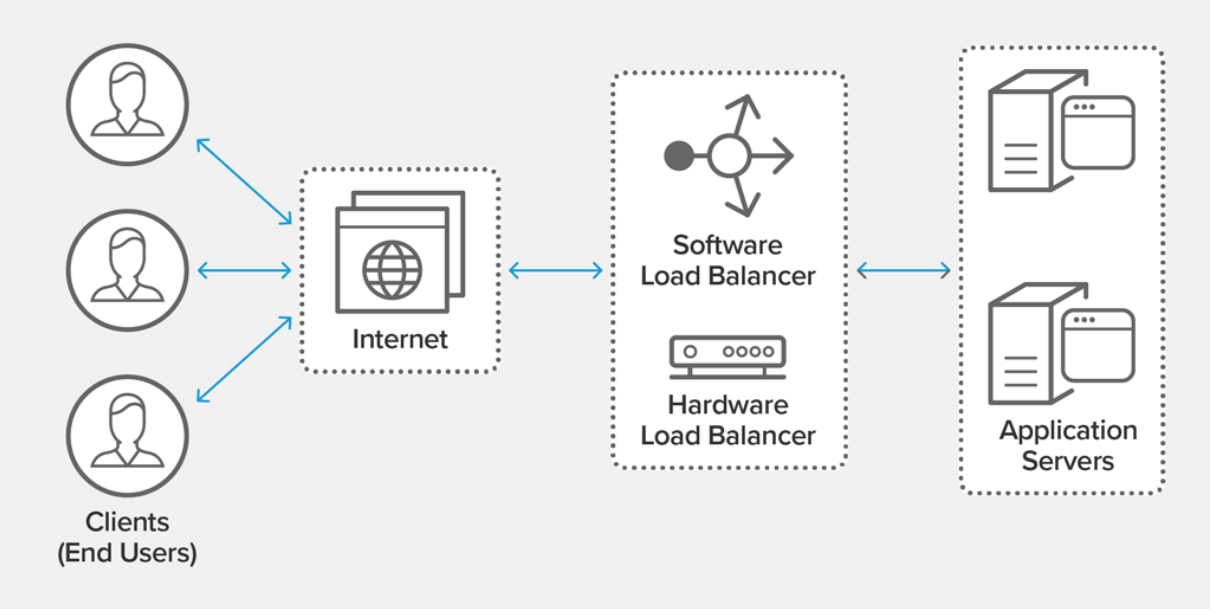
Tăng tính bảo mật cho hệ thống: thông thường khi người dùng gửi yêu cầu dịch vụ đến hệ thống, yêu cầu đó sẽ được xử lý trên bộ cân bằng tải, sau đó thành phần cân bằng tải mới chuyển tiếp các yêu cầu cho các máy chủ bên trong. Như vậy có thể ngăn chặn người dùng giao tiếp trực tiếp với các máy chủ, ẩn các thông tin và cấu trúc mạng nội bộ, ngăn ngừa các cuộc tấn công trên mạng hoặc các dịch vụ không liên quan đang hoạt động trên các cổng khác.

Khả năng mở rộng hệ thống: sự tăng vọt đột biến bất thường của lưu lượng truy cập gây ảnh hưởng rất lớn đến hiệu suất và tốc độ xử lý của hệ thống máy chủ,

cân bằng tải cung cấp khả năng thêm nhiều server hơn vào nhóm để quản lý các lượng request ngày càng tăng.

a) Cân bằng tải phần cứng và cân bằng tải phần mềm.

Bộ cân bằng tải phần cứng là một thiết bị phần cứng có hệ điều hành chuyên biệt phân phối lưu lượng ứng dụng web trên một cụm máy chủ ứng dụng. Để đảm bảo hiệu suất tối ưu, bộ cân bằng tải phần cứng phân phối lưu lượng truy cập theo các quy tắc tùy chỉnh để các máy chủ ứng dụng không bị quá tải.



Hình 1.12 Bộ cân bằng tải phần cứng và phần mềm.

Theo kiến trúc truyền thống, bộ cân bằng tải phần cứng và máy chủ ứng dụng được triển khai trong các trung tâm dữ liệu tại chỗ và số lượng bộ cân bằng tải phụ thuộc vào lượng lưu lượng truy cập cao điểm dự kiến. Bộ cân bằng tải thường được triển khai theo cặp trong trường hợp một bộ bị lỗi. Một bộ cân bằng tải phần cứng nằm giữa lưu lượng đến và các máy chủ nội bộ, về cơ bản hoạt động như một "cảnh sát lưu lượng". Khi khách hàng truy cập trang web, trước tiên chúng được gửi đến bộ cân bằng tải, sau đó chuyển hướng khách hàng đến các máy chủ khác nhau. Hầu hết các doanh nghiệp cũng triển khai bộ cân bằng tải và máy chủ ở nhiều vị trí, một quy trình được gọi là cân bằng tải máy chủ toàn cầu (GSLB). GSLB không chỉ cung cấp thời gian phản hồi được tối ưu hóa mà còn đảm bảo tính sẵn sàng cao trong các tình huống khôi phục sau thảm họa. Nếu một trung tâm dữ liệu bị lỗi, hệ thống GSLB có

thể chuyển hướng lưu lượng mạng đến các trang web có sẵn khác, giảm thiểu tác động đến người dùng cuối.

Cân bằng tải phần mềm là cách quản trị viên định tuyến lưu lượng mạng đến các máy chủ khác nhau. Bộ cân bằng tải đánh giá các yêu cầu của khách hàng bằng cách kiểm tra các đặc điểm cấp ứng dụng (địa chỉ IP, tiêu đề HTTP và nội dung của yêu cầu). Sau đó, bộ cân bằng tải sẽ xem xét các máy chủ và xác định máy chủ nào sẽ gửi yêu cầu đến. Cân bằng tải phần mềm thường được cung cấp dưới dạng một chức năng của bộ điều khiển phân phối ứng dụng chạy trên máy chủ tiêu chuẩn hoặc máy ảo. Cân bằng tải phần mềm cung cấp chức năng tương tự của cân bằng tải phần cứng, nhưng nó không yêu cầu thiết bị cân bằng tải chuyên dụng. Phần mềm cân bằng tải có thể chạy trên máy chủ thông thường hoặc thậm chí là máy chủ ảo.

b) Cân bằng tải tĩnh.

Thuật toán cân bằng tải là "tĩnh" khi nó không tính đến trạng thái của hệ thống để phân phối các tác vụ. Qua đó, trạng thái hệ thống bao gồm các thước đo như mức tải (và đôi khi là quá tải) của một số bộ xử lý nhất định. Thay vào đó, các giả định về hệ thống tổng thể được đưa ra trước, chẳng hạn như thời gian đến và yêu cầu tài nguyên của các tác vụ đến. Ngoài ra, số lượng bộ xử lý, công suất và tốc độ truyền thông tương ứng của chúng cũng được biết đến. Do đó, cân bằng tải tĩnh nhằm mục đích liên kết một tập hợp các nhiệm vụ đã biết với các bộ xử lý có sẵn để giảm thiểu một chức năng hiệu suất nhất định. Bí quyết nằm ở khái niệm của chức năng hiệu suất này.

Các kỹ thuật cân bằng tải tĩnh thường được tập trung xung quanh một bộ định tuyến, phân phối tải và tối ưu hóa chức năng hoạt động. Việc giảm thiểu này có thể tính đến thông tin liên quan đến các nhiệm vụ được phân phối và tính toán thời gian thực hiện dự kiến.

Ưu điểm của thuật toán tĩnh là chúng dễ thiết lập và cực kỳ hiệu quả trong trường hợp các tác vụ khá thường xuyên (chẳng hạn như xử lý các yêu cầu HTTP từ một trang web). Tuy nhiên, vẫn có một số phương sai thống kê trong việc phân công nhiệm vụ có thể dẫn đến quá tải cho một số đơn vị tính toán.

Các thuật toán tĩnh thích hợp cho các hệ thống có sự biến thiên của tải. Trong thuật toán tĩnh, lưu lượng được phân chia đồng đều giữa các máy chủ. Thuật toán này yêu cầu một kiến thức về tài nguyên hệ thống, hiệu suất của bộ xử lý được xác định khi bắt đầu thực thi, do đó quyết định chuyển tải không phụ thuộc về trạng thái hiện tại của hệ thống. Tuy nhiên, tải trọng tĩnh các thuật toán cân bằng có một nhược điểm là các nhiệm vụ chỉ được gán cho bộ xử lý hoặc các máy sau khi nó được tạo và không thể chuyển các nhiệm vụ trong quá trình thực thi sang bất kỳ máy khác để cân bằng tải.

c) Cân bằng tải động.

Các thuật toán cân bằng tải động tính đến tải trọng hiện tại của từng đơn vị tính toán (còn gọi là các nút) trong hệ thống. Theo cách tiếp cận này, các tác vụ có thể được di chuyển động từ một nút quá tải sang một nút thiếu tải để nhận được quá trình xử lý nhanh hơn. Mặc dù các thuật toán này phức tạp hơn nhiều để thiết kế, nhưng chúng có thể tạo ra kết quả tuyệt vời, đặc biệt, khi thời gian thực hiện thay đổi rất nhiều từ nhiệm vụ này sang nhiệm vụ khác.

Kiến trúc cân bằng tải động có thể mang tính mô-đun hơn vì không bắt buộc phải có một nút cụ thể dành riêng cho việc phân phối công việc. Khi các tác vụ được chỉ định duy nhất cho một bộ xử lý theo trạng thái của nó tại một thời điểm nhất định, đó là nhiệm vụ duy nhất. Mặt khác, nếu các nhiệm vụ có thể được phân phối lại vĩnh viễn theo trạng thái của hệ thống và sự phát triển của nó, thì điều này được gọi là phân công động. Rõ ràng, thuật toán cân bằng tải yêu cầu quá nhiều thông tin liên lạc để đạt được các quyết định của nó có nguy cơ làm chậm quá trình giải quyết vấn đề tổng thể.

Trong thuật toán động, máy chủ chịu tải ít nhất và có độ trễ đường truyền phù hợp trong toàn bộ mạng hoặc hệ thống được tìm kiếm và ưu tiên để cân bằng tải. Đối với giao tiếp thời gian thực với mạng là cần thiết có thể tăng lưu lượng truy cập trong hệ thống. Đây, trạng thái hiện tại của hệ thống được sử dụng để đưa ra quyết định quản lý tải. Các thuật toán động phản ứng với hệ thống hiện tại thực tế trạng thái trong việc đưa ra các quyết định chuyển tải. Kể từ khi trạng thái hiện tại của hệ thống

được sử dụng để đưa ra các quyết định cân bằng tải động, các quy trình được phép di chuyển từ một máy được sử dụng quá mức động đến một máy chưa được sử dụng trong thời gian thực.

Thuật ngữ "phân tán động" sử dụng trong luận văn được hiểu như là cách thức phân tán xử lý các tác vụ trong một hệ thống một cách động nhằm tăng tốc độ xử lý và tránh việc tắc nghẽn dữ liệu, quá tải hệ thống.

Kết luận Chương 1

Trong khuôn khổ của luận văn, học viên tập trung nghiên cứu và áp dụng hướng cân bằng tải động để áp dụng cho xử lý phân tán trong IoT Analyzer nhằm nâng cao hiệu quả việc phân tích và phát hiện mã độc IoT botnet.

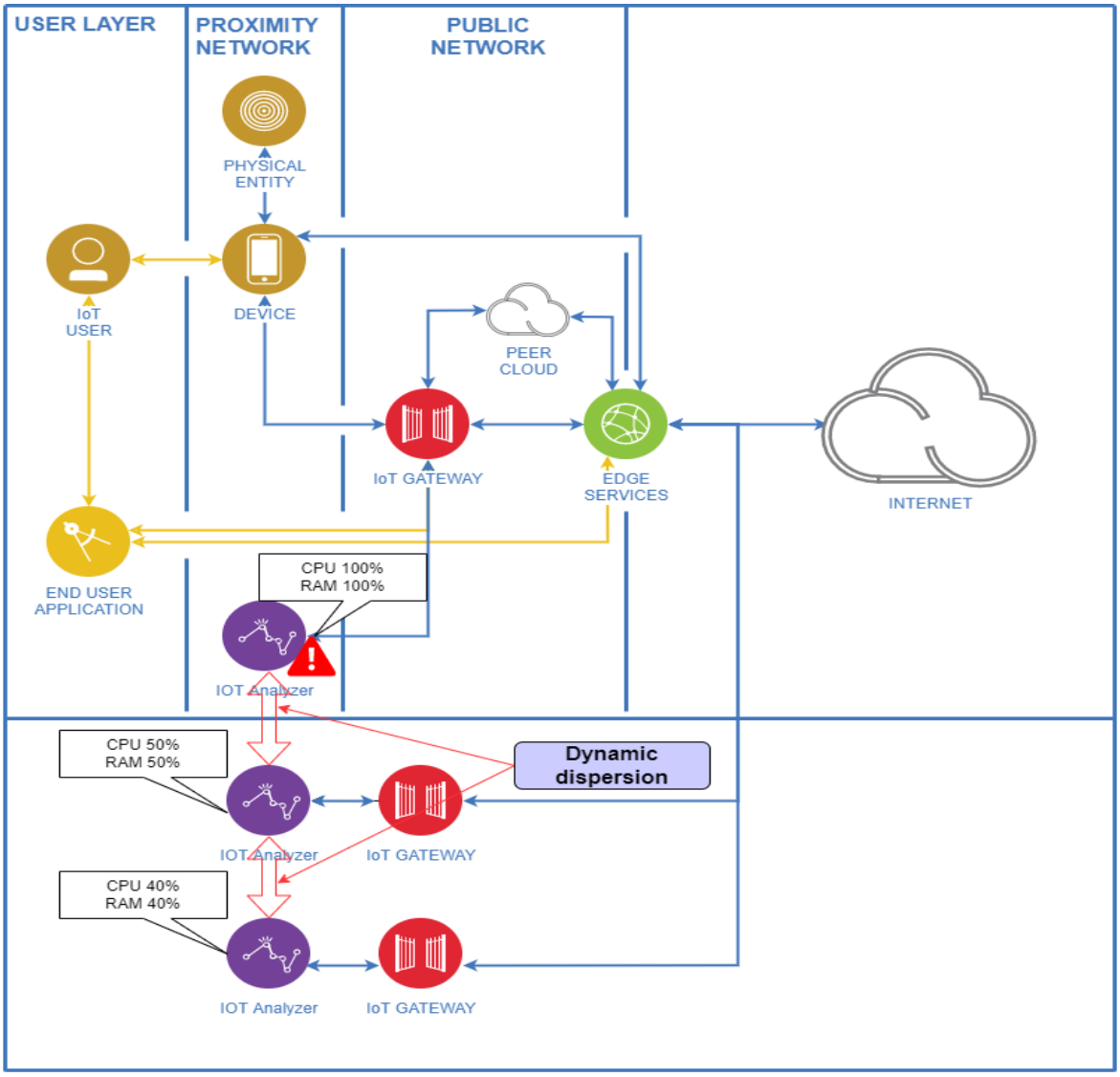
CHƯƠNG 2. ỨNG DỤNG GIẢI PHÁP PHÂN TÁN ĐỘNG TRONG HỆ THỐNG PHÂN TÍCH MÃ ĐỘC IOT BOTNET.

Trong chương 2, học viên sẽ trình bày về bài toán phân tán động trong hệ thống phân tích mã động IoT botnet nhằm tăng cường hiệu năng xử lý của toàn hệ thống. Từ những kiến thức quan trọng về cân bằng xử lý, học viên đưa ra phân tích, thiết kế và xây dựng mô hình xử lý bài toán sử dụng Apache Kafka và KSQL.

2.1. Phát biểu bài toán

Tại môi trường thực tế, số lượng thiết bị IoT là rất lớn và việc phân tích khối lượng dữ liệu cho toàn bộ các thiết bị IoT đòi hỏi một hệ thống được tối ưu tốt và có khả năng cân bằng năng lực xử lý. Với kịch bản một thiết bị IoT Analyzer quản lý và thu thập dữ liệu từ nhiều thiết bị IoT, khi IoT Analyzer bị quá tải thì sẽ cần một cơ chế để xử lý vấn đề này. Bài toán học viên đặt ra là làm thế nào để IoT Analyzer bị quá tải có thể phân tán các tác vụ cho một thiết bị IoT Analyzer khác với mục đích làm cho hệ thống không bị tắc nghẽn và gián đoạn.

Đầu vào của bài toán sẽ là các dữ liệu của thiết bị IoT được gửi cho IoT Analyzer trong kịch bản một thiết bị IoT Analyzer bị quá tải. Đầu ra của bài toán sẽ là việc phân chia tải cho một thiết bị IoT Analyzer khác. Yêu cầu của bài toán sẽ cần một cơ chế phân tán tác vụ và đảm bảo xử lý được một lượng lớn dữ liệu từ các thiết bị IoT. Mô hình bài toán được minh hoạ tại hình 2.1.



Hình 2.1. Bài toán phân tán xử lý tác vụ trong kịch bản một thiết bị IoT Analyzer quá tải.

2.1.1 Phân tích bài toán.

Trong phần phân tích bài toán, học viên đưa ra một số nhiệm vụ cần phải giải quyết trong bài toán, từ đó đưa ra các yêu cầu cụ thể để tìm kiếm giải pháp phù hợp.

Đầu tiên, phải xem xét đến đầu vào của bài toán là kịch bản khi một thiết bị IoT Analyzer bị quá tải. Thuật ngữ "quá tải" trong khuôn khổ bài toán sẽ được hiểu là thiết bị IoT Analyzer vượt một ngưỡng tải về % tải về CPU và % bộ nhớ RAM sử dụng. Do đó giải pháp đưa ra cần có phân hệ giám sát, thu thập thông tin của các thiết

bị IoT Analyzer. Việc xem xét một thiết bị IoT Analyzer là bị quá tải được định nghĩa bởi quản trị viên.

Tiếp theo, việc xử lý một thiết bị IoT Analyzer bị quá tải sẽ được xử lý như thế nào. Có hai cách tiếp cận để phân tán năng lực xử lý giữa các IoT Analyzer là phân tán tĩnh và phân tán động. Phân tán tĩnh là phương pháp xử lý khi không tính đến trạng thái tải hiện tại của hệ thống để phân phối tác vụ. Thay vì đó, phân tán tĩnh đưa ra các giải định về hệ thống trước và phân chia tải theo các thuật toán nhất định. Cách tiếp cận khác là phân tán động, hệ thống sẽ truy xuất các thông tin về tải tại các nút tính toán, từ đó đưa ra các quyết định phân chia tải cho các nút ít tải từ một nút quá tải. Phân tán động cho khả năng phân chia tác vụ một cách mềm dẻo hơn và có thể tăng tốc quá trình xử lý. Tuy nhiên giải pháp phân tán động sẽ phức tạp hơn do cần các thông tin đo lường hệ thống cũng như thuật toán chia sẻ tải.

Giải pháp cho việc phân chia tải yêu cầu phải có khả năng kích hoạt quá trình gửi dữ liệu giữa các thiết bị IoT Analyzer khi bị quá tải. Quá trình kích hoạt này được thực hiện tự động khi ngưỡng giới hạn về sử dụng tài nguyên trên một thiết bị IoT Analyzer được kích hoạt. Song song với đó, cách thức truyền và nhận dữ liệu giữa các IoT Analyzer cũng cần được xem xét để không làm chậm quá trình xử lý trong hệ thống. Bảng 2.1 mô tả các yêu cầu khi phân tích bài toán.

Bảng 2.1 Các yêu cầu bài toán.

Các bước	Yêu cầu
Xử lý đầu vào	- Giám sát, thu thập thông tin tải của thiết bị IoT Analyzer. - Định nghĩa ngưỡng quá tải của thiết bị IoT Analyzer. - Tự động kích hoạt khi bị quá tải.
Xử lý dữ liệu	- Phân tán tĩnh hoặc phân tán động - Hỗ trợ gửi dữ liệu giữa các thiết bị IoT Analyzer - Hỗ trợ kích hoạt và dừng gửi dữ liệu giữa các thiết bị IoT Analyzer
Đầu ra	- Hệ thống không bị gián đoạn. - Tăng năng lực xử lý của hệ thống

2.1.2 Khảo sát, đánh giá các giải pháp.

Các thuật toán hỗ trợ cân bằng năng lực xử lý động trên thế giới đã và đang được nghiên cứu và ứng dụng cụ thể. Qua khảo sát các năm gần đây, một số giải pháp công trình khoa học đã được công bố được trình bày trong phần dưới.

a) Thuật toán Biased Random Sampling

Cách tiếp cận này [19] sử dụng một đồ thị ảo để cân bằng tải trong hệ thống dựa trên lưới. Trong cách tiếp cận này, mỗi nút trong biểu đồ đại diện cho một máy chủ mức độ trong của mỗi nút đại diện cho số lượng tài nguyên miễn phí của nó. Mạng được xây dựng bằng cách tạo liên kết giữa các nút được chọn ngẫu nhiên. Khi một nút nhận được một công việc mới, mức độ trong giảm xuống để cho thấy rằng các tài nguyên khả dụng bị giảm đi và khi nút hoàn thành công việc, mức độ trong của nó được tăng lên để cho thấy rằng hiện có nhiều tài nguyên hơn.

Quá trình tăng và giảm được thực hiện thông qua lấy mẫu ngẫu nhiên. Mọi thứ sẽ bắt đầu tại nút nhận công việc và tiếp tục với người hàng xóm được chọn ngẫu nhiên của nó. Hiệu quả của việc phân bổ tải tăng lên khi chiều dài đi bộ tăng lên, w. Thực nghiệm [19] cho thấy rằng ngưỡng 'w' hiệu quả là khoảng log (n) bước, trong đó n là kích thước mạng.

Khi một nút nhận được một công việc, nó sẽ thực thi nó nếu độ dài hiện tại của công việc lớn hơn hoặc bằng ngưỡng độ dài. Nếu không, giá trị w của công việc được tăng lên và nó được gửi đến một hàng xóm ngẫu nhiên, do đó tiếp tục phương pháp lấy mẫu ngẫu nhiên. Phiên bản cải tiến của thuật toán này [20] phân bổ yêu cầu mới cho nút được tải ít nhất trong bước ngẫu nhiên. Ở đây số lượng tài nguyên miễn phí đóng vai trò là tham số để cân bằng tải. Do đó, yêu cầu mới được phân bổ cho nút có số lượng tài nguyên miễn phí tối đa.

b) Thuật toán Active Clustering

Kiến trúc được đề xuất là phi tập trung, do đó loại bỏ vấn đề tắc nghẽn trong mạng. Nó cũng tính đến vấn đề cân bằng tải trên một mạng không đồng nhất. Mạng được chia thành các cụm bằng cách sử dụng thuật toán phân cụm sao cho mọi nút (chủ) trong mạng thuộc về chính xác một cụm. Các nút mới tham gia vào mạng được

chỉ định một cụm hiện có hoặc một cụm mới trong trường hợp cụm bị chiếm hoàn toàn dựa trên chiến lược phân nhóm được sử dụng. Mỗi cụm có ít nhất một nút Giao tiếp Liên cụm (ICC). Do đó, khả năng mở rộng của kiến trúc phụ thuộc vào kỹ thuật phân cụm. Việc phân cụm được thực hiện trong các giai đoạn ban đầu trong khi mạng đang được khởi tạo. Các loại nút và chức năng của chúng như sau.

Slave là phần tử tính toán của mạng. Việc xử lý các tác vụ được thực hiện trong các phần tử này. Mỗi Slave được kết nối trực tiếp với chính xác máy chủ Master. Vì hệ thống ban đầu đã coi là một mạng không đồng nhất, nên khả năng tính toán của các Slave có thể khác nhau trên toàn mạng. Thuật toán sử dụng các thuộc tính sau của các Slave để quyết định yếu tố tối ưu của việc tính toán tác vụ: bộ nhớ khả dụng, bộ nhớ khả dụng và băng thông khả dụng. Slave cập nhật định kỳ tổng thể của nó với các giá trị gần đây nhất của các tham số trên. Máy chủ Matser là một máy tính toán quyết định chính sách cân bằng tải của các tác vụ giữa các nút phụ và chọn một nút phụ để thực hiện các tác vụ. Nút chính duy trì một bảng phân phối tải giữa các Slave. Mỗi mục của bảng mô tả tải trên phụ tương ứng. Bảng này được cập nhật mỗi khi cái chủ giao một nhiệm vụ mới cho một phụ hoặc khi một tác vụ cũ được hoàn thành trên một số nút phụ và kết quả được gửi lại cho cái chính. Mỗi cụm có ít nhất một Liên

Nút Giao tiếp cụm (ICC), được định nghĩa là nút chính nằm cách một nút ICC của cụm khác một bước nhảy. Thuật toán cân bằng tải này chia thành hai phần tùy thuộc vào loại nút và giao tiếp giữa chúng. Các bộ phận này được gọi là: (1) Phân bổ tải giữa các bản chính; và (2) Phân phối tải từ chủ đến phụ.

c) Thuật toán Honey Bee Foraging

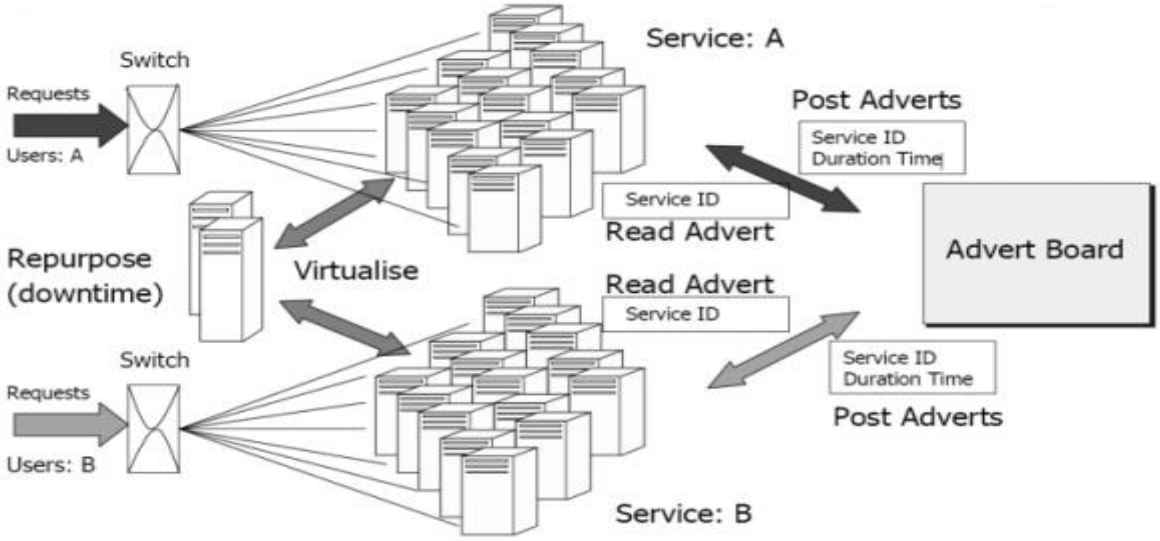
Trong [21], [22],cân bằng tải được trình bày như một nghiên cứu điển hình, phù hợp với cho kiến trúc dịch vụ chung. Mô phỏng thuật toán cân bằng tải dựa trên tổ ong,theo nghiên cứu [23] đã được sử dụng ở lớp ứng dụng để phân bổ tài nguyên tối ưu. Công việc đã kiểm tra việc phân bổ các máy chủ trong SOA quy mô lớn; phân bổ máy chủ cho các ứng dụng Web để tối đa hóa dung lượng và hiệu quả sử dụng. Các nhà nghiên cứu thường coi một kịch bản như vậy có thể áp dụng cao cho Điện toán đám mây và việc áp dụng mô hình trả tiền cho mỗi lần sử dụng (điện toán tiện ích).

Thuật toán được sử dụng trong bài báo này tuân theo thuật toán [23] để điều phối các máy chủ lưu trữ Dịch vụ Web. Đây là một trong số các ứng dụng được lấy cảm hứng từ hành vi được cho là của một đàn ong mật kiếm ăn và thu hoạch thức ăn.

Kỹ thuật lấy cảm hứng từ sinh học này hiện được sử dụng như một thuật toán tìm kiếm trong nhiều ứng dụng máy tính khác nhau. Như vậy, khi áp dụng để cân bằng tải; khi nhu cầu về Dịch vụ Web dao động, nên phân bổ động các máy chủ để điều chỉnh hệ thống theo nhu cầu. Kỹ thuật cân bằng tải dựa trên honeybee được coi là sử dụng một tập hợp các máy chủ được sắp xếp thành các máy chủ ảo, mỗi máy phục vụ một hàng đợi yêu cầu dịch vụ ảo. Lợi nhuận được tính theo mỗi máy chủ phục vụ yêu cầu từ hàng đợi - đại diện cho thước đo chất lượng của những con ong. Thước đo lợi nhuận này được tính toán dựa trên "chi phí" của máy chủ trong việc phục vụ máy chủ ảo (ví dụ: thời gian CPU đã sử dụng), có tính đến "phần thưởng" do máy chủ ảo phân bổ cho thời gian của nó (mô hình doanh thu loại dịch vụ). Tương tự, "sàn nhảy" của tổ ong được phản ánh trong một bảng cân bằng tải không gian chia sẻ được phân phối. Bảng cân bằng tải này cũng được sử dụng để truyền đạt khái niệm về lợi nhuận thuộc địa toàn cầu (trên mỗi máy chủ ảo); một thước đo về mức độ hiệu quả của các nguồn lực đang được sử dụng. Tùy thuộc vào các yêu cầu của miền cụ thể, đây có thể là tổng hợp đơn giản của thời gian nhàn rỗi (lãng phí) CPU của máy chủ riêng lẻ hoặc có thể bao gồm các chỉ số khác, chẳng hạn như độ dài hàng đợi dịch vụ trung bình. Tổng quan cấp cao về sự tương tác giữa các máy chủ (được nhóm thành hai máy chủ "ảo"), hàng đợi dịch vụ mà chúng phục vụ và bảng cân bằng tải được chia sẻ được hiển thị bên dưới trong Hình 2.2 [23].

Máy chủ ảo, hoạt động cân bằng tải và mỗi máy chủ đảm nhận một vai trò ong cụ thể với xác suất **px** hoặc **pr**. Những giá trị này được sử dụng để bắt chước thuộc địa ong mật, theo đó một số lượng ong nhất định được giữ lại làm thức ăn chăn nuôi - để khám phá (**px**); chứ không phải là người thu hoạch - để khai thác các nguồn hiện có. Máy chủ đáp ứng yêu cầu thành công sẽ đăng trên bảng cân bằng tải với xác suất **pr**. Máy chủ có thể chọn ngẫu nhiên hàng đợi của máy chủ ảo với xác suất **px** (khám phá), nếu không sẽ kiểm tra. Tóm lại, các máy chủ nhàn rỗi (ong chờ đợi) tuân theo một trong hai kiểu hành vi: máy chủ đọc bảng cân bằng tải sẽ theo dõi thông báo đã

chọn, sau đó phục vụ yêu cầu; do đó bắt chước hành vi thu hoạch. Một máy chủ không đọc bảng cân bằng tải sẽ quay lại hành vi kiếm ăn; phục vụ yêu cầu hàng đợi của máy chủ ảo ngẫu nhiên.



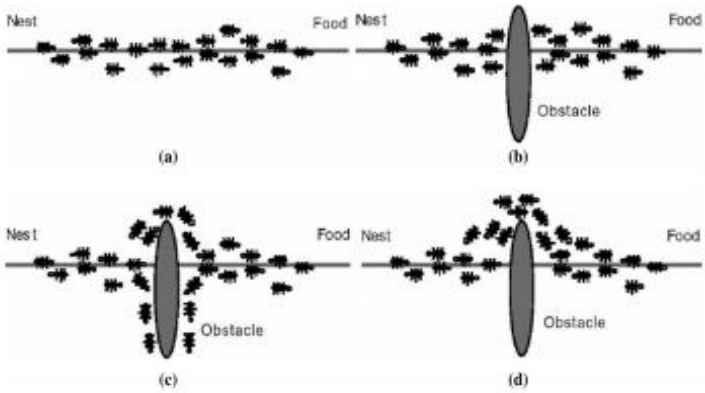
Hình 2.2. Máy chủ ảo và bảng thông báo

Một máy chủ thực thi sẽ hoàn thành yêu cầu và tính toán lợi nhuận của máy chủ ảo chỉ được phục vụ. Máy chủ đã hoàn thành (tức là ong quay lại) ảnh hưởng đến hành vi của hệ thống bằng cách so sánh lợi nhuận được tính toán của nó với lợi nhuận thuộc địa trên bảng cân bằng tải, sau đó điều chỉnh **px** (kiểm soát tỷ lệ khám phá / khai thác) và lợi nhuận thuộc địa cho phù hợp. Nếu lợi nhuận được tính là cao, thì máy chủ quay trở lại máy chủ ảo hiện tại; đăng một thông báo cho nó theo xác suất **pr**. Nếu lợi nhuận thấp, thì máy chủ sẽ quay lại trạng thái nhàn rỗi/chờ đợi được mô tả ở trên. Ban đầu, mọi máy chủ đều bắt đầu với hành vi khám phá/kiếm ăn và khi các yêu cầu được phục vụ, hành vi hướng dẫn thông báo/waggle-dance bắt đầu xuất hiện. Với một phương pháp tính toán lợi nhuận mạnh mẽ, mô hình hành vi này cung cấp một cơ chế giao tiếp toàn cầu và phân tán; đảm bảo các máy chủ ảo "sinh lời" có vẻ hấp dẫn và được phân bổ cho các máy chủ có sẵn.

d) Thuật toán Ant Colony Optimization (ACO)

ACO được lấy cảm hứng từ các đàn kiến làm việc cùng nhau trong hành vi kiếm ăn. Trên thực tế, những con kiến thật đã truyền cảm hứng cho nhiều nhà nghiên cứu trong công việc của họ [24], [25], và cách tiếp cận kiến đã được nhiều nhà nghiên

cứu sử dụng để giải quyết vấn đề trong nhiều lĩnh vực khác nhau. Cách tiếp cận này được gọi theo tên của nguồn cảm hứng ACO. Kiến làm việc cùng nhau để tìm kiếm nguồn thức ăn mới và đồng thời sử dụng nguồn thức ăn hiện có để chuyển thức ăn về tổ.



Hình 2.3 .Mô hình đàn kiến hoạt động

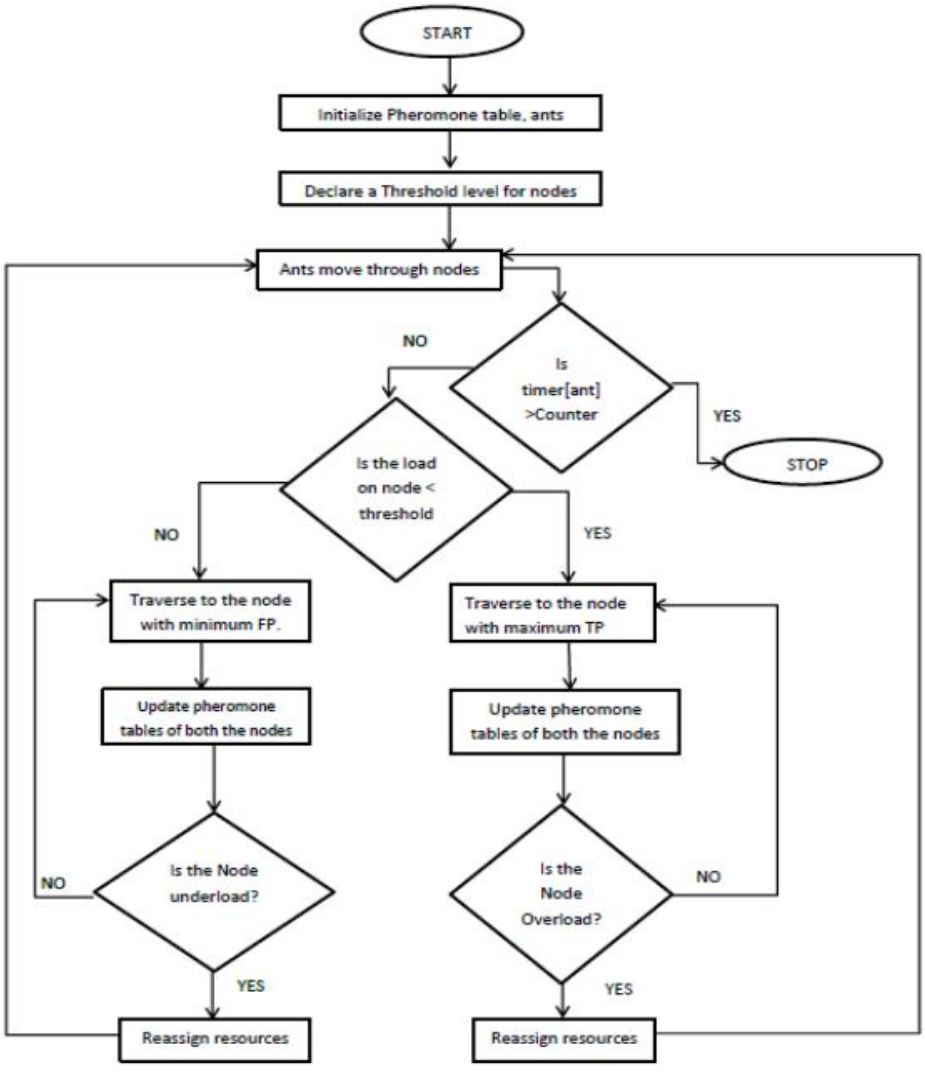
Khi phát hiện ra rằng những con kiến để lại dấu vết pheromone khi di chuyển từ nút này sang nút khác. Bằng cách đi theo các đường mòn pheromone, kiến sau đó đã đến nguồn thức ăn. Cường độ của pheromone có thể thay đổi tùy thuộc vào nhiều yếu tố khác nhau như chất lượng nguồn thức ăn, khoảng cách thức ăn,… Kiến sử dụng những đường mòn pheromone này để chọn nút tiếp theo. Những con kiến thậm chí có thể sửa đổi đường đi khi gặp bất kỳ chướng ngại vật nào trên đường đi của chúng. Hình 2.3 đưa ra một ý tưởng ngắn gọn về kịch bản này. Hiện tượng này của kiến đã được sử dụng trong nhiều thuật toán để tối ưu hóa nơi các kiến đi theo nhau thông qua một mạng lưới các đường dẫn pheromone. Những con kiến khi di chuyển từ nút này sang nút khác sẽ cập nhật dấu vết pheromone của con đường đó, vì vậy một con đường trở nên khả thi hơn nếu có nhiều con kiến đi qua nó. Những con đường có cường độ pheromone cao nhất có khoảng cách ngắn nhất giữa điểm và nguồn thức ăn tốt nhất.

Các chuyển động của những con kiến này cập nhật một cách độc lập một bộ giải pháp. Di chuyển của kiến trong hệ thống này nói chung có hai kiểu:

1) Di chuyển về phía trước - Trong kiểu di chuyển này, kiến đi chuyển để lấy thức ăn hoặc tìm kiếm nguồn thức ăn.

2) Di chuyển lùi - Trong kiểu di chuyển này, kiến sau khi nhặt thức ăn từ nguồn thức ăn sẽ bay về tổ để dự trữ thức ăn của chúng.

ACO là một thuật toán duy nhất vì một số lý do như giải pháp tối ưu được xây dựng không phải bởi một thực thể đơn lẻ mà là nhiều thực thể khác nhau, đi qua chiều dài và chiều rộng của mạng và sau đó những giải pháp này riêng lẻ xây dựng dựa trên một giải pháp. Nhiều nhà nghiên cứu [26] để cải thiện kết quả cũng đã ứng biến theo hiện tượng cập nhật pheromone của ACO. Nó đã được các nhà nghiên cứu sử dụng để cải thiện các nhiệm vụ khác nhau như lập lịch tác vụ hoặc tối ưu hóa trong mạng vệ tinh [27].



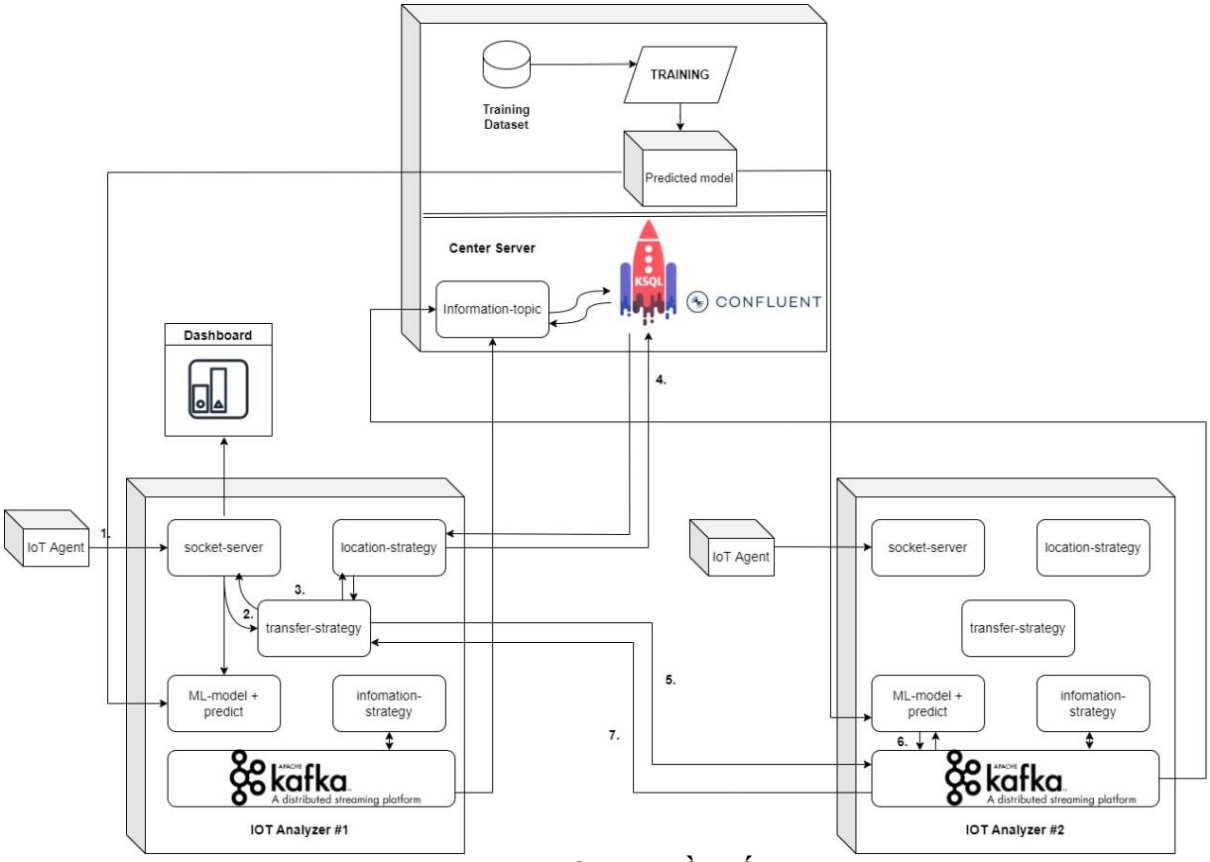
Hình 2.4. Thuật toán ACO

Các thuật toán trình bày bên trên đều giải quyết được bài toán cân bằng tải với các cách thức hoạt động và xử lý khác nhau. Tuy nhiên các thuật toán này khi áp dụng vào bài toán phân tán động áp dụng trong hệ thống phân tích và phát hiện mã độc IoT

thì chưa giải quyết triệt để việc phân tán động. Trong phần tiếp theo, học viên đề xuất một mô hình trong đó, khi một nút bị quá tải nó sẽ là nút chủ và chủ động chọn các nút có tải thấp hơn để chia sẻ tác vụ.

2.2. Mô hình đề xuất

Để đảm bảo các yêu cầu bài toán, học viên đưa ra mô hình đề xuất được minh hoạ tại hình 2.5. Mô hình gồm 2 thành phần chính gồm Center Server và các IoT Analyzer. Các thành phần và hoạt động của hệ thống được mô tả chi tiết ở phần sau.



Hình 2.5 Mô hình đề xuất

2.2.1. Các thành phần trong mô hình

Từ những yêu cầu chức năng trong mô hình, có thể liệt kê ra các hoạt động chính của mô hình và các chủ thể đối tượng người dùng tương ứng. Chi tiết các hoạt động chính được trình bày trong Bảng 2.2. Các đối tượng chính tham gia vào hệ thống này gồm:

- Thiết bị phân tích cục bộ (IoT Analyzer): Thiết bị mã hóa dữ liệu, kiểm tra dữ liệu được mã hóa, gửi dữ liệu được mã hóa về máy chủ trung tâm. Thiết bị IoT Analyzer có chức năng cập nhật tri thức an toàn thông tin, xử lý dữ liệu lớn và huấn luyện mô hình học máy/học sâu phát hiện tấn công mạng.
- Máy chủ giám sát trung tâm (Center Server): Máy chủ trung tâm thực hiện quản trị hệ thống, giám sát tập trung, phân phối tài nguyên hệ thống,.
- Dữ liệu hành vi luồng mạng thu thập từ các thiết bị IoT cỡ nhỏ đã được chuẩn hoá.

Bảng 2.2 Danh sách các hoạt động chính của hệ thống

STT	Hệ thống tương tác	Đối tượng	Sự kiện, hành vi
1	IoT Analyzer	Thông tin tài nguyên hệ thống	Thực hiện việc theo dõi giám sát, lấy dữ liệu, thông tin về tài nguyên của hệ thống tại máy chủ IoT Analyzer và thực hiện kiểm tra dữ liệu trên thiết bị IoT có bị nhiễm mã độc hay không.
2	Center Server	Cấu hình giới hạn tài nguyên	Thực thi cầu hình giới hạn tài nguyên của hệ thống tại máy chủ Center Server
3	Center Server	Danh sách các máy chủ cục bộ và thông tin tài nguyên hệ thống của từng máy	Thực thi cập nhật danh sách tài nguyên hệ thống của từng máy chủ cục bộ IoT Analyzer tại máy chủ trung tâm Center Server
4	IoT Analyzer	Thông tin tài nguyên hệ thống	Thực thi gửi yêu cầu hỗ trợ và thông tin tài nguyên hệ thống tới máy chủ Center Server
5	Center Server	Yêu cầu hỗ trợ quá tải	Thực hiện lấy thông tin máy chủ cục bộ hỗ trợ gửi về IoT Analyzer bị quá tải
6	IoT Analyzer	Thông tin máy chủ hỗ trợ, dữ liệu luồng mạng và hành vi	Gửi dữ liệu luồng mạng và hành vi tới máy chủ hỗ trợ dự trên thông tin mà máy chủ trung tâm đã cung cấp.

STT	Hệ thống tương tác	Đối tượng	Sự kiện, hành vi
7	IoT Analyzer	Thông tin máy chủ bị quá tải, dữ liệu luồng mạng và hành vi được cung cập	Thực hiện việc nhận dữ liệu luồng mạng và hành vi, xử lý dữ liệu đã nhận và gửi dữ liệu đã xử lý tới máy chủ bị quá tải

2.2.2. Đặc tả yêu cầu chi tiết

a) Yêu cầu chức năng

1) Chức năng "Cấu hình giới hạn tài nguyên hệ thống tại máy chủ trung tâm"

- Mô tả: Quản trị viên kích hoạt chức năng cấu hình giới hạn tài nguyên hệ thống tại IoT Analyzer

- Dữ liệu đầu vào: Dữ liệu, cấu hình giới hạn tài nguyên hệ thống tại IoT Analyzer

- Xử lý dữ liệu: Cấu hình giới hạn tài nguyên hệ thống tại IoT Analyzer trước khi thực hiện thu thập tài nguyên hệ thống. Khi ngưỡng tài nguyên được cấu hình, hệ thống sẽ thực hiện giám sát và đưa ra kết quả quá tải hay không dựa trên ngưỡng.

$$\left(CPU_{usage} < threshold \middle| RAM_{usage} < threshold\right)$$

- Kết quả đầu ra: Hệ thống lưu cấu hình giới hạn tài nguyên hệ thống tại IoT Analyzer thành công.

2) Chức năng "Thu thập dữ liệu tài nguyên hệ thống tại máy chủ cục bộ"

- Mô tả: Máy chủ trung tâm thu thập dữ liệu tài nguyên hệ thống tại máy chủ cục bộ.

- Dữ liệu đầu vào: lệnh để thu thập dữ liệu tài nguyên hệ thống tại máy chủ cục bộ.

- Xử lý dữ liệu: Hệ thống chạy lệnh để thu thập dữ liệu tài nguyên hệ thống tại máy chủ cục bộ tương ứng.

- Kết quả đầu ra: Thu thập và gửi thông tin tài nguyên hệ thống tới máy chủ trung tâm thành công.

- Thuật toán:

Khởi tạo cấu hình ngưỡng tài nguyên hệ thống: $CPU_{threshold}, RAM_{threshold}$
While True:
    Hàm xử lý thu thập hostname (get_hostname()).
    Hàm xử lý thu thập IP (get_private_IP()).
    Hàm xử lý thu thập địa chỉ MAC (get_mac_address()).

Hàm xử lý thu thập % CPU sử dụng (get_cpu_percent()).
Hàm xử lý thu thập % RAM sử dụng (get_ram_used()).
Nếu % CPU sử dụng và % RAM sử dụng nằm trong ngưỡng:
Trạng thái hệ thống = OK (System status = OK)
Ngược lại: Trạng thái hệ thống = NOK (System status = NOK)
Gửi các thông tin hệ thống cho máy chủ trung tâm gồm (host name, IP, MAC, %CPU, % RAM, trạng thái hệ thống).
Chờ 1 giây.

3) Chức năng "Gửi yêu cầu hỗ trợ từ máy chủ bị quá tải về máy chủ trung tâm"

- Mô tả: Máy chủ trung tâm khóa dữ liệu về máy chủ cục bộ.

- Dữ liệu đầu vào: Khóa đã được điểm tra và được đánh giá tốt.

- Xử lý dữ liệu: Gửi yêu cầu hỗ trợ từ máy chủ bị quá tải về máy chủ trung tâm.

- Kết quả đầu ra: Gửi yêu cầu hỗ trợ từ máy chủ bị quá tải về máy chủ trung tâm thành công.

Thuật toán:

Khởi tạo cấu hình ngưỡng tài nguyên hệ thống: $CPU_{threshold}, RAM_{threshold}$
Nếu có yêu cầu xử lý dữ liệu từ IoT Agent (Task assignment == True):
Nếu trạng thái hệ thống bị quá tải (System status == NOK):
Gửi yêu cầu hỗ trợ đến máy chủ trung tâm (Check available support node).
Ngược lại: Xử lý tại máy chủ IoT Analyzer cục bộ (Predict Request to ML-model)

4) Chức năng "Cập nhật danh sách tài nguyên hệ thống tại máy chủ trung tâm"

- Mô tả: Cập nhật danh sách tài nguyên hệ thống tại máy chủ trung tâm.

- Dữ liệu đầu vào: tài nguyên hệ thống.

- Xử lý dữ liệu: Hệ thống cho phép tự động cập nhật danh sách tài nguyên hệ thống tại máy chủ trung tâm.

- Kết quả đầu ra: Cập nhật danh sách tài nguyên hệ thống tại máy chủ trung tâm thành công.

- Thuật toán:

Khởi tạo: Các IoT Analyzer gửi thông tin tài nguyên cho máy chủ trung tâm.
Thực hiện cập nhật thông tin tài nguyên hệ thống theo chu kỳ.
Nếu có yêu cầu xử lý quá tải từ IoT Analyzer:
    Gửi danh sách thông tin tài nguyên hệ thống cho IoT Analyzer

5) Chức năng "Xử lý yêu cầu hỗ trợ quá tải tại IoT Analyzer"

- Mô tả: Xử lý yêu cầu hỗ trợ quá tải.
- Dữ liệu đầu vào: yêu cầu hỗ trợ quá.
- Xử lý dữ liệu: Xử lý yêu cầu hộ trợ quá tải.
- Kết quả đầu ra: Xử lý yêu cầu hộ trợ quá tải thành công.
- Thuật toán:

Khởi tạo: Trạng thái hệ thống
Nếu nhận được yêu cầu hỗ trợ xử lý quá tải từ nút khác:
    Kiểm tra trạng thái hệ thống cục bộ.
    Nếu trạng thái hệ thống cục bộ = OK:
        Nhận thông tin chưa được xử lý từ nút yêu cầu hỗ trợ.
        Gọi mô đun học máy thực hiện phân tích và phát hiện mã độc xử lý dữ liệu.
        Trả kết quả về nút bị quá tải.
    Ngược lại: Từ chối yêu cầu xử lý.

6) Chức năng "Gửi dữ liệu chưa xử lý tới máy chủ hỗ trợ"

- Mô tả: Gửi dữ liệu chưa xử lý tới máy chủ hỗ trợ.
- Dữ liệu đầu vào: Dữ liệu luồng mạng và hành vi chưa được xử lý; Thông tin máy chủ hỗ trợ.
- Xử lý dữ liệu: Hệ thống cho phép gửi dữ liệu chưa xử lý tới máy chủ hỗ trợ.
- Kết quả đầu ra: Hệ thống gửi dữ liệu chưa xử lý tới máy chủ hỗ trợ thành công.

7) Chức năng "Xử lý dữ liệu nhận từ máy chủ bị quá tải"

- Mô tả: Xử lý dữ liệu nhận từ máy chủ bị quá tải.
- Dữ liệu đầu vào: Dữ liệu luồng mạng và hành vi chưa được xử lý; Thông tin máy chủ bị quá tải.

- Xử lý dữ liệu: Xử lý dữ liệu nhận từ máy chủ bị quá tải.
- Kết quả đầu ra: Hệ thống xử lý dữ liệu nhận từ máy chủ bị quá tải thành công

b) Yêu cầu phi chức năng

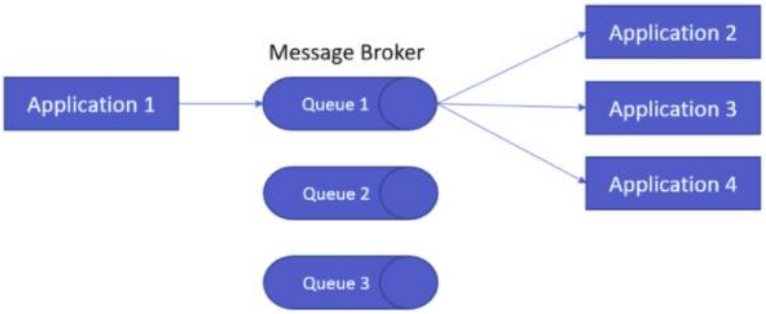
Yêu cầu đặt ra là phải phát triển được hệ thống thỏa mãn các yêu cầu phi chức năng sau:

- Hỗ trợ đọc các tệp dữ liệu lưu trữ theo bảng mã Unicode.
- Hỗ trợ đa nền tảng hệ điều hành phổ biến dành cho máy chủ phân tích bao gồm: Windows, Linux, Unix,…
- Hỗ trợ khả năng sao lưu, khôi phục dữ liệu lưu trữ khi gặp sự cố.

2.3. Ứng dụng Kafka và KSQL trong xây dựng giải pháp xử lý phân tán động

2.3.1 Khảo sát một số công cụ message broker có sẵn.

Để thực hiện được công việc xử lý bản tin (message), học viên đưa ra một số tiêu chí lựa chọn công cụ trong luận văn như sau: Mã nguồn mở, khả năng lưu trữ và xử lý dữ liệu lớn tốt, có khả năng chịu lỗi tốt, có khả năng mở rộng, hiệu năng tốt.



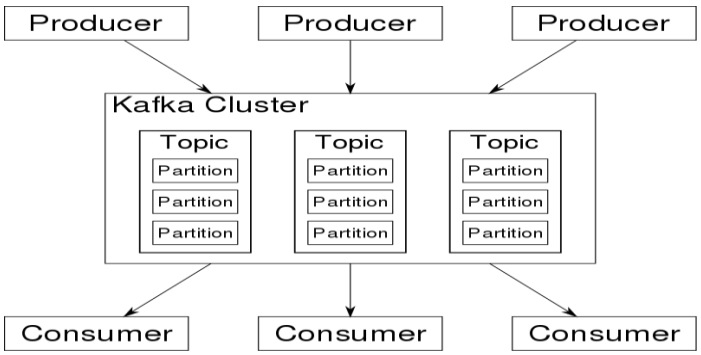
Trên thế giới hiện nay có rất nhiều công cụ phân phối bản tin (message broker) được áp dụng rất nhiều trong các doanh nghiệp và sản phẩm thương mại. Một số công cụ có thể kể đến mà Redis, RabbitMQ, Kafka. Trong khuôn khổ luận văn, học viên đánh giá 3 công cụ này và chọn một công cụ phù hợp nhất để ứng dụng vào giải pháp cân bằng tải động.

Công cụ	Khả năng mở rộng	Bộ nhớ	Phương thức giao tiếp
RabbitMQ	Dựa trên cấu hình và tài nguyên, khả năng xử lý khoảng 50 nghìn message/giây.	Hỗ trợ cả lưu tin nhắn lâu dài (persistent message) và nhất thời (transient message)	Hỗ trợ cả one-to-one và one-to-many
Redis	Có thể xử lý lên đến 1 triệu message/giây.	Dữ liệu được lưu tại bộ nhớ trong (in-memory database).	Hỗ trợ cả one-to-one và one-to-many
Kafka	Có thể xử lý lên đến 1 triệu message/giây.	Hỗ trợ cả lưu tin nhắn lâu dài (persistent message) và nhất thời (transient message)	Hỗ trợ one-to-many

Dựa vào bảng so sánh bên trên cho thấy công cụ Kafka là một công cụ phù hợp nhất so với các công cụ còn lại. Kafka có khả năng xử lý dữ liệu lớn tốt, khả năng lưu trữ tin nhắn lâu dài, ngoài ra Kafka có khả năng hỗ trợ rất tốt xử lý bất đồng bộ với ngôn ngữ python. Một điểm lợi thế của Kafka đó là hỗ trợ KSQL, một công cụ hỗ trợ truy vấn thông tin luồng dữ liệu (data stream) tương tự như truy vấn cơ sở dữ liệu quan hệ. Do đó học viên lựa chọn công cụ Kafka và KSQL để ứng dụng giải pháp phân tán động trong luận văn. Phần tiếp theo sẽ mô tả chi tiết công cụ Kafka và KSQL.

2.3.2 Lựa chọn công cụ Kafka và công cụ KSQL.

Nền tảng trao đổi thông điệp Apache Kafka: Apache Kafka [28] là một nền tảng xử lý stream mã nguồn mở được phát triển bởi Apache Software Foundation được viết bằng ngôn ngữ lập trình Scala và Java. Kafka cho hệ thống truyền tải dữ liệu phân tán. Kafka lưu trữ thông điệp trong các chủ đề (topic) được phân vùng và sao chép qua nhiều broker trong một cụm (Cluster). Publisher (nguồn gửi thông tin) gửi thông điệp đến các topic mà nguồn nhận thông tin (consumer) đã đăng ký để đọc thông điệp. Kafka hoạt động tốt như một sự thay thế cho một hệ thống gửi thông tin (broker) truyền thống hơn. So với hầu hết các hệ thống broker, Kafka có thông lượng tốt hơn, khả năng phân vùng dữ liệu lưu trữ thông điệp, tích hợp khả năng sao chép và khôi phục thông điệp khi xảy ra lỗi đường truyền, khiến nó trở thành một giải pháp tốt cho các ứng dụng xử lý thông điệp quy mô lớn.



Hình 2.6. Kiến trúc truyền tải thông tin của Apache Kafka

Thông báo là mảng byte (với String, JSON và Avro là các định dạng phổ biến nhất). Nếu một thông điệp có khóa, Kafka đảm bảo rằng tất cả các thông điệp có cùng khóa đều nằm trong cùng một phân vùng. Consumer có thể được gộp chung vào một nhóm consumer. Mỗi consumer trong một nhóm consumer sẽ đọc tin nhắn từ một tập hợp con duy nhất của các phân vùng trong mỗi topic mà chúng đăng ký. Mỗi thông điệp được gửi đến một consumer trong nhóm và tất cả các thông điệp có cùng khóa đến cùng một consumer. Kafka không theo dõi tin nhắn nào đã được đọc bởi từng consumer. Kafka lưu giữ tất cả các tin nhắn trong một khoảng thời gian hữu hạn và consumer có trách nhiệm theo dõi vị trí của chúng trên mỗi topic.

Kafka có các chức năng như:

- Xử lý lượng lớn dữ liệu;
- Xử lý dữ liệu từ nhiều nguồn và nhiều định dạng;
- Có khả năng hoạt động một cách liền mạch với các phần mềm khác;
- Xử lý dữ liệu thời gian thực ngay khi tiếp nhận;
- Cho phép bên nhận và bên gửi hoạt động độc lập tại các thời điểm khác nhau;
- Hỗ trợ các ngôn ngữ lập trình như Java, C hoặc Python. Hỗ trợ các ngôn ngữ front-end như HTML, CSS và JavaScript;
- Có thể hoạt động dựa trên điện toán đám mây;
- Có độ trễ nhỏ giữa dữ liệu đầu vào và phản hồi;
- Cho phép dữ liệu và thông điệp tích hợp trực tiếp vào các ứng dụng sử dụng API;

- Phạm vi tích hợp tiêu chuẩn cho các nhà cung cấp dữ liệu bên thứ nhất, thứ hai và thứ ba;
- Khả năng kiểm soát quyền truy cập tệp/thư mục của người dùng hoặc nhóm, quyền chia sẻ bên ngoài, chính sách chỉnh sửa, giới hạn vị trí thiết bị, chia sẻ theo thiết bị…;
- Phát hiện các bất thường về chức năng, khả năng tiếp cận của người dùng, luồng lưu lượng truy cập và giả mạo;
- Chủ động theo dõi trạng thái của các trạm làm việc tại chỗ hoặc từ xa;

Ưu điểm của Kafka bao gồm:

- Khả năng truyền thông điệp nhanh và đáng tin cậy;
- Hỗ trợ xử lý dữ liệu song song;
- Phân chia dữ liệu hợp lý;
- Có khả năng mở rộng quy mô xử lý dữ liệu cho nhiều người dùng;
- Có cộng đồng hỗ trợ kỹ thuật đông đảo và cập nhật;
- Miễn phí (Open-source).

Nhược điểm của Kafka cụ thể gồm:

- Giao diện người dùng chưa được tốt, vẫn dưới dạng cửa sổ dòng lệnh;
- Quá trình cài đặt và sử dụng có thể hơi phức tạp với người dùng không chuyên.

KSQL là một cơ sở dữ liệu được xây dựng có mục đích cho các ứng dụng xử lý luồng. Nó hợp nhất nhiều thành phần được tìm thấy trong hầu hết mọi kiến trúc xử lý luồng. Điều đó rất quan trọng vì hầu hết tất cả các kiến trúc phát trực tuyến ngày nay đều là các giải pháp từng phần được kết hợp với nhau từ các dự án khác nhau. Ở mức tối thiểu, cần một hệ thống con để thu thập các sự kiện từ các nguồn dữ liệu hiện có, một hệ thống khác để lưu trữ chúng, một hệ thống khác để xử lý chúng và một hệ thống khác để phục vụ các truy vấn chống lại các dữ liệu tổng hợp. Việc tích hợp từng hệ thống con có thể khó khăn.

KSQL cho phép xử lý và phân tích dữ liệu truyền trực tuyến theo thời gian thực có trong nền tảng Apache Kafka. Nói cách khác, KSQL cung cấp một khung

tương tác để thực hiện các hoạt động Xử lý Luồng như Tổng hợp dữ liệu, Lọc, Kết hợp, Sessionization, Windowing và hơn thế nữa. Hơn nữa, thực thi truy vấn bằng KSQL tương tự như chạy truy vấn SQL trong Cơ sở dữ liệu quan hệ. Các lệnh chính của KSQL như SELECT, LIMIT, JOIN và WHERE giống như các lệnh có trong SQL. Kafka được sử dụng trong các ứng dụng khác nhau để xử lý hoặc phân tích dữ liệu truyền trực tuyến, bao gồm giám sát liên tục, phân tích phát trực tuyến theo thời gian thực, tích hợp dữ liệu trực tuyến, phát hiện bất thường, thăm dò dữ liệu và lọc tùy ý.

Các trường hợp phù hợp để ứng dụng KSQL:

- Giám sát thời gian thực và phân tích thời gian thực: KSQL có thể được sử dụng để theo dõi thời gian thực và phân tích thời gian thực trên dữ liệu.
- Bảo mật và phát hiện bất thường: Trong KSQL, có thể xác định các truy vấn tổng hợp và phát hiện bất thường trên các luồng thời gian thực.
- Tích hợp dữ liệu trực tuyến: Sử dụng KSQL, các nhóm hệ thống có thể hợp nhất các luồng dữ liệu khác nhau trong thời gian thực.

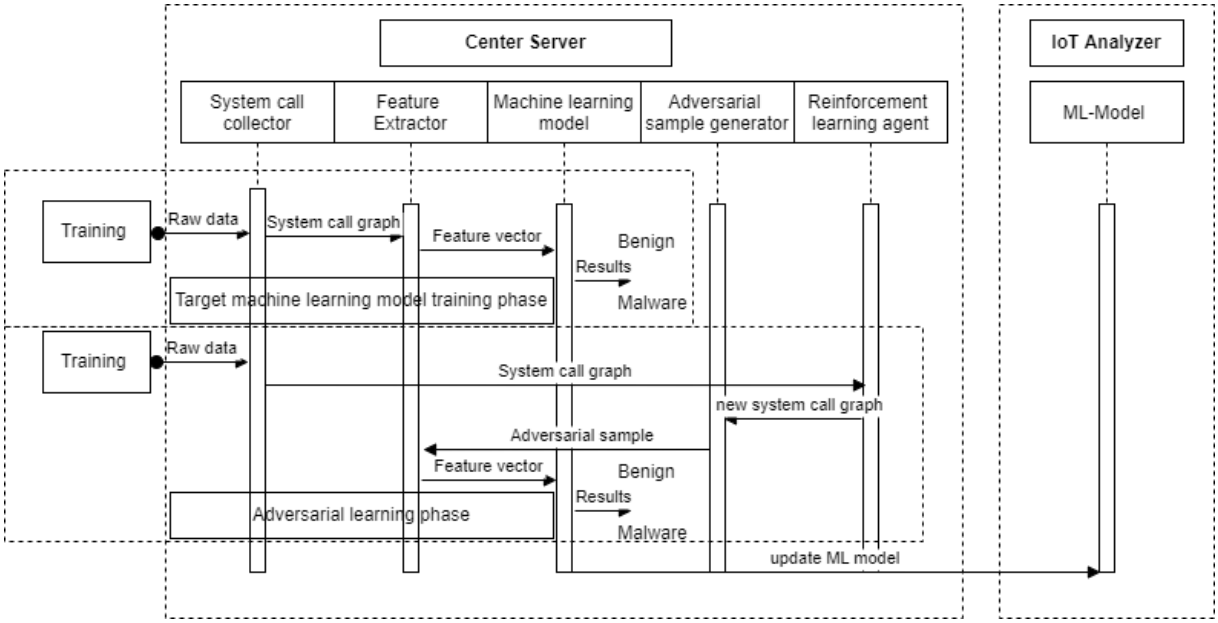
Hoạt động của KSQL như sau: KSQL chứa quy trình máy chủ thực thi các truy vấn của Kafka. Một tập hợp các quy trình này có thể được chạy dưới dạng Kafka Cluster. Nếu muốn tăng thêm sức mạnh xử lý, chỉ cần khởi chạy các phiên bản khác của Máy chủ KSQL này. Ngoài ra, các máy chủ này có khả năng chịu lỗi, do đó, nếu một trong số chúng bất thường, máy chủ còn lại sẽ tiếp quản công việc của chúng. Thủ tục truy vấn rất đơn giản và nó được thực hiện với giao diện dòng lệnh KSQL (CLI), giao diện này sẽ gửi truy vấn đến cụm qua API REST. Dòng lệnh KSQL là một công cụ hữu ích để kiểm tra tất cả các chủ đề Kafka hiện có, tạo luồng và bảng cũng như kiểm tra tiến trình của các truy vấn này.

2.3.3 Hoạt động của mô hình.

a) Máy chủ trung tâm

Máy chủ trung tâm có hai nhiệm vụ chính là (1) thực hiện đào tạo mô hình học máy để phát hiện mã độc IoT Botnet và (2) theo dõi, lưu trữ thông tin tải của tất cả hệ thống.

Mô đun đào tạo mô hình học máy: Mô hình phân tích và phát hiện mã độc được sử dụng tại phần 1.2.3. Tập dataset gồm 1107 mẫu trong đó có 507 mẫu mã độc và 600 mẫu lành tính được thu thập tại IoTPOT. Kết quả thực nghiệm cho thấy mô hình có khả năng phát hiện mã độc lên đến 88.94% và có khả năng phát hiện các mã độc zero-day. Sau khi mô hình được huấn luyện sẽ được lưu lại và gửi về các IoT Analyzer qua quản trị viên. Tại các IoT Analyzer sẽ có khả năng dựa vào mô hình đã huấn luyện để phát hiện các mã độc dựa trên thông tin IoT Agent gửi lên.



Hình 2.7 Mô hình huấn luyện học máy phân tích và phát hiện mã độc IoT botnet

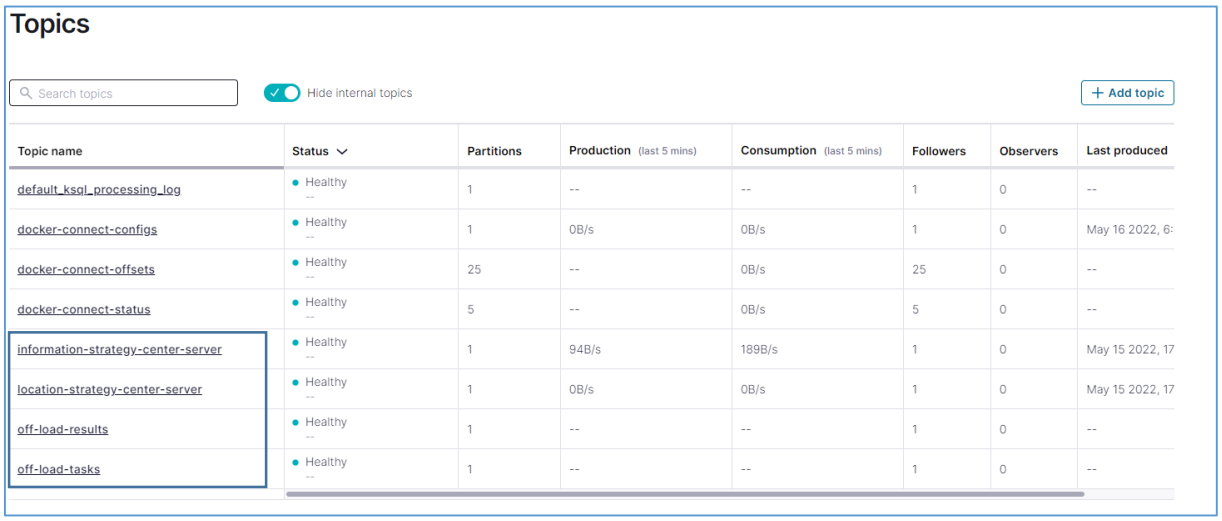
Bảng 2.3 Kết quả đánh giá mô hình phát hiện và phân tích mã độc.

Machine learning classifier	ACC	TPR	FPR	Precision	F1-Score	AUC	Detect new malwares
NB	0.7101	0.5198	0.05	0.9291	0.6667	0.8649	25/107
DT	0.8378	0.7665	0.0722	0.9305	0.8406	0.8471	63/107
KNN	0.8673	0.7797	0.0222	0.9779	0.8676	0.9358	65/107
SVM	0.8894	0.8282	0.0333	0.9691	0.8931	0.9738	73/107
RF	0.8550	0.7753	0.0444	0.9565	0.8564	0.9549	66/107

Mô đun theo dõi, lưu trữ thông tin tải của hệ thống được coi là trái tim của hệ thống. Mô đun này được sử dụng platform Confluent [29] trong đó tích hợp sẵn công cụ Kafka và KSQL. Tại đây, các Kafka broker thu thập các thông tin trạng thái tải của hệ thống gửi lên thông qua Kafka producer và lưu lại trong KSQL nhằm phục vụ việc lưu trữ thông tin. Thông tin được lưu tại Kafka Topic, từ đó khi có bất kỳ yêu cầu xử lý từ các IoT Analyzer, các message trong đó có thông tin trạng thái tải của các IoT Analyzer ở trạng thái rảnh rỗi sẽ được trả về.

Trên máy chủ trung tâm, các Kafka broker và Kafka topic như sau:

- Kafka broker: Thực hiện tạo các topic chứa thông tin, tại các IoT Analyzer sẽ là Kafka consumer và Kafka producer gửi và nhận thông tin từ Kafka broker.
- Kafka topic: Information-topic được tạo để lưu trữ thông tin tải của toàn bộ hệ thống. Kafka topic được minh họa như ở hình :



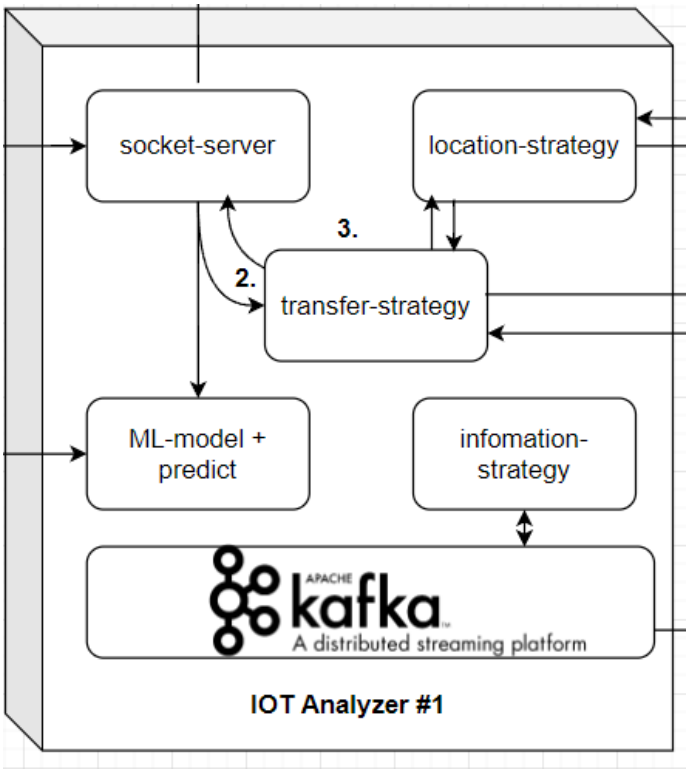
Hình 2.8 Kafka Topic tại máy chủ trung tâm.

b) Thiết bị phân tích IoT Analyzer

Cấu trúc của IoT Analyzer được minh họa tại hình 2.9 gồm 5 thành phần chính:

- socket-server chịu trách nhiệm nhận dữ liệu chưa được xử lý từ IoT Agent và lưu lại tại IoT Analyzer. Sau khi dữ liệu được phân tích và đánh giá, kết quả sẽ được socket-server trả về một máy chủ Dashboard.

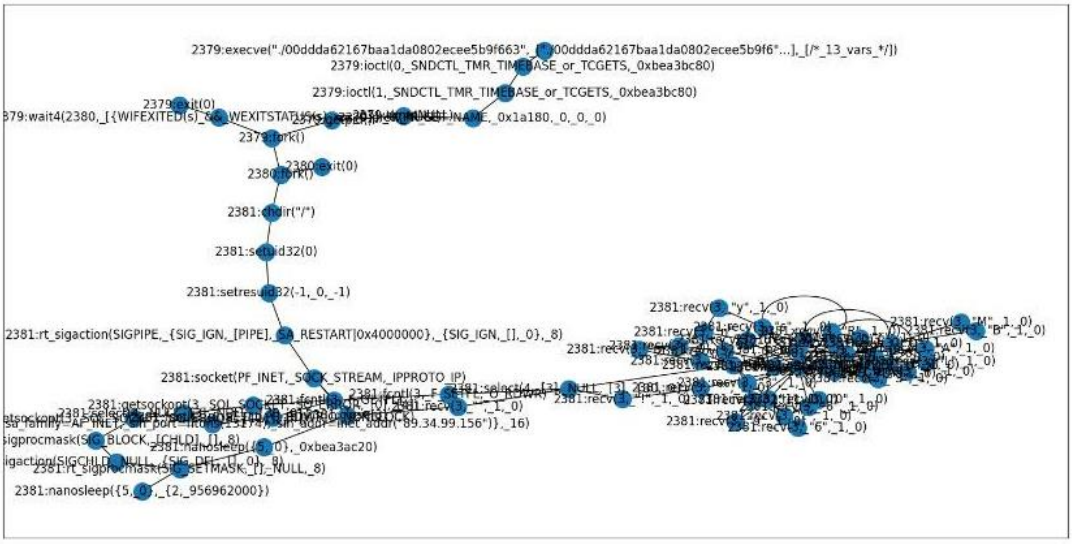
- ML-model + predict: là mô hình học máy xử lý các dữ liệu nhận được từ IoT Agent, phân tích và đưa ra đánh giá dữ liệu có phải mã độc hay không. Mô hình học máy này được cập nhật từ phía máy chủ trung tâm.
- information-strategy là mô đun thu thập thông tin trạng thái của hệ thống, sau đó sử dụng Kafka producer truyền message gồm các thông tin hệ thống của nó lên máy chủ trung tâm.
- location-strategy là mô đun xử lý truy vấn lên máy chủ tập trung để yêu cầu chia sẻ tác vụ khi bị quá tải. Mô đun này cũng sẽ chịu trách nhiệm chọn ra IoT Analyzer nào để đẩy các tác vụ sang.
- transfer-strategy là mô đun truyền nhận dữ liệu chưa được xử lý từ một IoT Analyzer quá tải sang một IoT Analyzer khác được chọn từ location-stategy.



Hình 2.9 Các mô đun trong IoT Analyzer

c) Thiết bị IoT Agent

Thiết bị IoT Agent là các thiết bị IoT có khả năng thu thập thông tin hoạt động của nó và gửi lên cho IoT Analyzer. Các thông tin được sử dụng là các lệnh gọi hệ thống (system call) được gửi lên cho IoT Analyzer bằng một TCP socket.



Hình 2.10 Đồ thị lệnh gọi hệ thống của một malware

d) Cơ chế cân bằng tải.

Thuật toán cân bằng tải bao gồm một số thành phần tương tác theo nhiều cách khác nhau để phân phối lại các công việc đã gửi của người dùng giữa các nút của hệ thống phân tán. Mục tiêu là cải thiện hiệu suất hệ thống bằng cách chia sẻ toàn bộ sức mạnh tính toán có sẵn. Có thể xác định ba thành phần chính: chính sách thu thập thông tin, chính sách kiểm soát chuyển giao và chính sách kiểm soát địa điểm/đàm phán.

- Đo tải cục bộ: Tại mỗi nút, một cơ chế phải được cung cấp để đưa ra ước tính tốt về tải cục bộ hiện tại. Các lựa chọn thay thế đo tải cục bộ được mô tả trong tài liệu bao gồm: chỉ số "tải trung bình" được cung cấp bởi lệnh thử viện psutil trong Python.

- Chính sách thông tin: Thành phần này chịu trách nhiệm trao đổi và duy trì thông tin cục bộ hoặc nhóm nút khác như mức tải, khối lượng công việc hoặc tải trung bình trên toàn bộ hệ thống. Nó cũng chịu trách nhiệm về tần suất cập nhật thông tin trạng thái, cách thức trao đổi thông tin này giữa các nút khác nhau, số lượng nút tham gia trao đổi và lượng thông tin cung cấp để thực hiện phân tán tác vụ. Nó phải duy trì thông tin nhất quán về trạng thái toàn cầu tại các điểm kiểm soát được phân phối. Có hai loại kỹ thuật cập nhật thông tin toàn cầu: cập nhật định kỳ các bảng thông tin sử

dụng cơ chế phát tin hoặc cập nhật thông tin theo yêu cầu dựa trên kỹ thuật thăm dò. Một ví dụ của kỹ thuật thứ hai là việc truy vấn các nút lân cận khi nút đó không hoạt động. Trong khuôn khổ luận văn, mô hình đề xuất sẽ sử dụng cơ chế cập nhật thông tin thứ hai.

- Chính sách chuyển giao: Thành phần này quyết định thời điểm có lợi khi chuyển một quy trình từ khối lượng công việc cục bộ và chọn quy trình nào để chuyển các tác vụ. Nút quá tải chọn một nút chưa quá tải để chuyển, dựa trên thông tin cục bộ, thông tin từ xa được duy trì cục bộ hoặc có được trong quá trình thương lượng với các nút khác. Chính sách chuyển giao cũng chịu trách nhiệm yêu cầu chuyển công việc từ các nút khác khi nút cục bộ sắp không hoạt động. Chính sách chuyển giao là thành phần tối thiểu cần thiết để triển khai chiến lược cân bằng tải, các bước của chính sách chuyển giao:

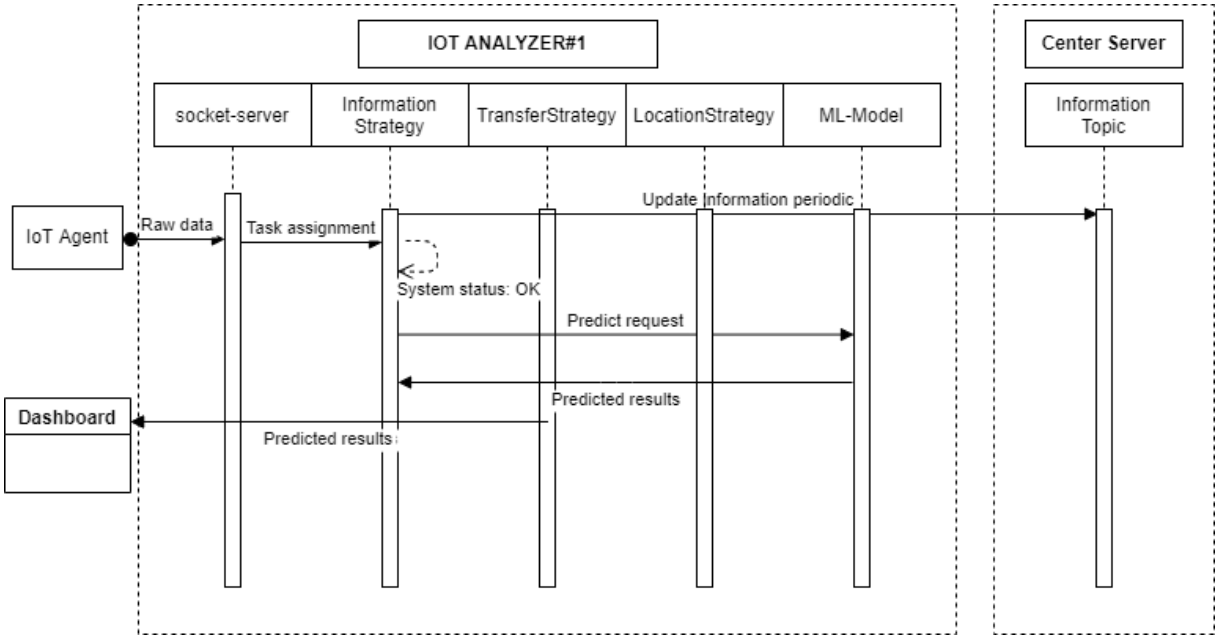
  + Nút khởi động quá trình cân bằng tải? Có ba cách để chỉ định điều kiện của nút bắt đầu quá trình phân phối lại tải: nút gửi (quá tải) cố gắng đẩy các tác vụ cho nút người nhận hoặc nút chưa quá tải cố gắng kéo công việc hoặc chuyển đổi động giữa hai công việc đẩy hoặc kéo bất cứ khi nào phù hợp. Trường hợp này được gọi là cân bằng tải bắt đầu đối xứng. Trong mô hình đề xuất, việc nút bắt đầu quá trình phân phối tác vụ là nút quá tải.

  + Khi nào thì thuận lợi để chuyển giao hoặc nhận các quy trình? (1) Các sự kiện kích hoạt cân bằng tải: Sự kiện kích hoạt phân phối lại tải sử dụng một chỉ báo quá tải. Chỉ báo này tương ứng với mức ngưỡng cho phép được quản trị viên cấu hình từ trước.

  + Hình thức chuyển giao: Một công việc có thể được phân bổ cho một nút từ xa trước khi nó bắt đầu thực hiện. Khi công việc được nút từ xa hoàn thành sẽ trả kết quả về nút bị quá tải.

- Chính sách đàm phán: Khi một nút quá tải quyết định rằng một nút khác là nút chuyển giao phù hợp (với ngưỡng tải được cấu hình/bộ nhớ RAM hoặc độ trễ mạng), nó sẽ tham gia vào một quá trình ghép nối. Quá trình này bao gồm việc tìm kiếm đối tác chuyển giao, chuyển các dữ liệu chưa được xử lý cho nút sẽ được phân

phối tác vụ. Quy tắc lựa chọn nút chuyển giao dựa trên chỉ báo tải được lưu trên máy chủ trung tâm. Kiểm tra nút có khả năng phân chia tác vụ là việc chọn nút có giá trị nhỏ nhất trong số một tập các nút khả dụng.



Hình 2.11 Lưu đồ hoạt động của hệ thống ở trạng thái trong tải

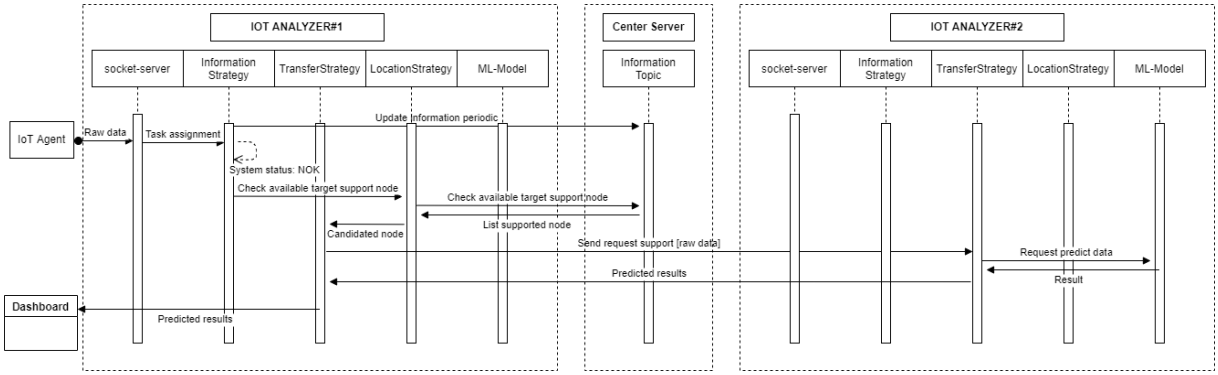
Khi IoT Analyzer hoạt động trong trạng thái không bị quá tải (load < threshold), luồng công việc được mô tả trong hình 2.11. Cụ thể các bước thực hiện như sau:

- Bước 1: IoT Analyzer cập nhật trạng thái hệ thống của nó thường xuyên cho máy chủ trung tâm qua Information Strategy
- Bước 2: IoT Agent gửi dữ liệu chưa được xử lý lên cho IoT Analyzer. Dữ liệu này là các tệp thực thi lệnh gọi hệ thống được xử lý thành các đồ thị. (system call graph).
- Bước 3: IoT Analyzer nhận dữ liệu bằng socket-server và thông báo rằng hệ thống cần xử lý dữ liệu.
- Bước 4: Information Strategy sẽ kiểm tra trạng thái hệ thống, kết quả trả về hệ thống đang ở trạng thái không bị quá tải. Điều kiện để hệ thống ở trạng thái không bị quá tải:

$$\left(CPU_{usage} < threshold \middle| RAM_{usage} < threshold\right)$$

Trong đó threshold = 90%.

- Bước 5: Mô đun học máy sẽ thực hiện phân tích và đưa ra dữ liệu nhận được có phải mã độc hay không và phản hồi lại.
- Bước 6: IoT Analyzer sẽ gửi kết quả đến cho Dashboard.



Hình 2.12 Lưu đồ hoạt động của hệ thống khi một node bị quá tải

Khi một nút bị quá tải (load>threshold), luồng công việc sẽ được thực hiện như sau: information-stategy chạy trên từng IoT Analyzer duy trì việc cập nhật tình trạng tài nguyên hệ thống trên máy chủ trung tâm.

- Bước 1: IoT Analyzer cập nhật trạng thái của nó định kỳ cho máy chủ trung tâm qua Information Strategy.
- Bước 2: IoT Agent gửi dữ liệu chưa được xử lý về nút IoT Analyzer. Information Strategy kiểm tra trạng thái hệ thống có bị quá tải hay không. Điều kiện để hệ thống không bị quá tải như sau:

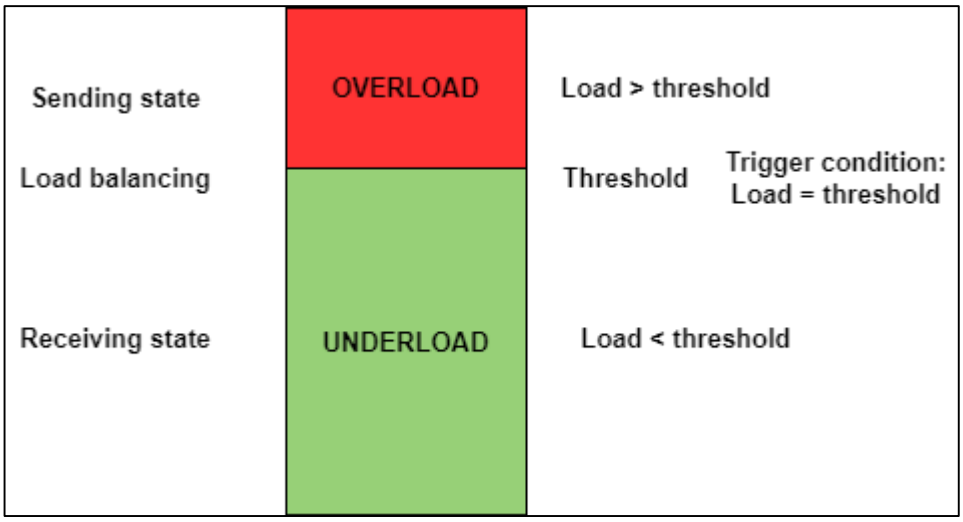
$$\left(CPU_{usage} < threshold \middle| RAM_{usage} < threshold\right)$$

Trong đó threshold = 90%.

- Bước 3: socket-server gọi location-stategy để tìm kiếm nút đáp ứng yêu cầu tính toán.
- Bước 4: location-strategy thực hiện tìm kiếm nút đủ điều kiện chuyển giao đáp ứng được nhu cầu tính toán (dựa trên RAM, CPU và độ trễ mạng). Location-strategy sử dụng KSQL API để truy vấn dữ liệu trong topic

information-stategy nằm trên máy chủ trung tâm và nhận về danh sách các nút có thể đáp ứng được.

- Bước 5: IoT Analyzer bị quá tải tiến hành lệnh ping tới các nút trong danh sách trên, nút có kết quả ping thấp nhất sẽ được chọn để phân phối tác vụ.
- Bước 6: transfer-stategy truyền thông tin từ nút quá tải sang nút từ xa sử dụng Kafka.
- Bước 7: Nút từ xa xử lý dữ liệu và tính toán, sau đó trả về nút quá tải.



Hình 2.13 Vùng tải và trạng thái cân bằng tải trên một nút IoT Analyzer

Kết luận chương 2.

Chương 2 học viên đã trình bày về bài toán phân tán động cần xử lý trong hệ thống phân tích mã độc IoT botnet, từ đó đưa ra những phân tích, yêu cầu của hệ thống. Học viên cũng đã đề xuất một mô hình phân tán động nhằm giải quyết bài toán nêu trên. Cụ thể trong chương 2 đã trình bày:

- Khảo sát, đánh giá các thuật toán hỗ trợ phân tán động và cân bằng tải động.
- Phân tích, thiết kế chức năng phân tán động giữa các thiết bị IoT Analyzer.
- Lựa chọn sử dụng các công nghệ mới để tích hợp giải pháp hỗ trợ cân bằng tải động và phân tán năng lực xử lý giữa các thiết bị IoT Analyzer.

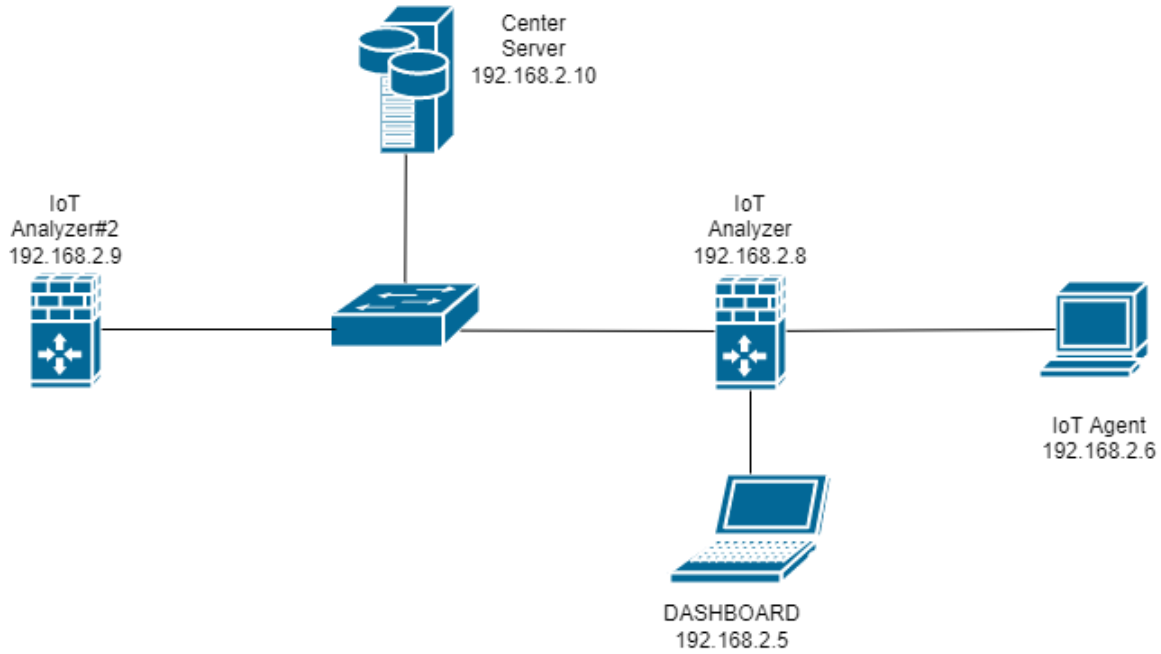
CHƯƠNG 3. THỬ NGHIỆM VÀ ĐÁNH GIÁ.

Trong chương 3, học viên sẽ trình bày về cách thức xây dựng môi trường thử nghiệm và kết quả đánh giá mô hình. Với mô hình đề xuất, hệ thống thử nghiệm sẽ được mô tả chi tiết trong chương này nhằm đánh giá chính xác năng lực của mô hình. Các kịch bản và thông số đánh giá cũng được định nghĩa để đưa ra ưu nhược điểm của mô hình đề xuất. Cuối cùng của chương sẽ trình bày kết quả thử nghiệm và kết luận.

3.1. Môi trường thử nghiệm

3.1.1 Hệ thống thử nghiệm

Hệ thống thử nghiệm bao gồm Máy chủ trung tâm, hai máy chủ đóng vai trò IoT Analyzer, 1 máy trạm đóng vai trò như IoT Agent và 1 máy trạm đóng vai trò như máy chủ Dashboard. Tất cả các máy đều sử dụng chung một hệ thống mạng và được kết nối với nhau thông qua một bộ chuyển mạch. Trên máy chủ trung tâm sẽ được cài đặt Confluent platform cho Kafka và KSQL, trên các máy IoT Analyzer được cài đặt Kafka để truyền các thông tin tải và dữ liệu. Mô hình thử nghiệm được minh hoạ trong hình 3.1. Bảng 3.1 mô tả chi tiết hệ thống thử nghiệm.



Hình 3.1 Mô hình thử nghiệm

Bảng 3.1 Thông tin chi tiết hệ thống thử nghiệm.

Server	Hardware	Repository	OS
Máy chủ trung tâm	CPU Intel Xeon E5-2689V1 @2.6GHz + 32GB RAM + GPU RX570 4GB GDDR5	Docker, Confluent, Docker-compose, Python, Java	Ubuntu 20.04 LTS
IoT Analyzer	Intel Core I5 8300H @2.3GHz + 16GB RAM + GPU GTX1050Ti 4GB GDDR5	Kafka, Python	Ubuntu 20.04 LTS
IoT Agent	Intel Core I5 8300H @2.3GHz + 16GB RAM + GPU GTX1050Ti 4GB GDDR5		Ubuntu 20.04 LTS

3.1.2. Quá trình triển khai giải pháp đề xuất

Tại máy chủ trung tâm: Cài đặt platform confluent trong đó tích hợp công cụ Kafka và KSQL.

Bước 1: Cài đặt công cụ Docker và Docker compose

Bước 2: Tạo file docker-compose.yml mô tả các container dịch vụ của Kafka.

Bước 3: Truy cập vào giao diện GUI và tạo các Topic.

Tại IoT Analyzer: Thực hiện chạy các module cho giải pháp cân bằng tải. Khi một IoT Agent gửi file đến IoT Analyzer, socket server sẽ nhận file và bắt đầu thực hiện xử lý.

Bước 1: Copy source code đến IoT Analyzer.

Đường dẫn source code: https://github.com/huunguyen95/distributed-model-thesis

Bước 2: Chạy module Information Strategy để gửi thông tin tài nguyen cho máy chủ trung tâm bằng câu lệnh: *"nohup python3 InformationStrategy.py &"*

Bước 3: Chạy module socket server bằng câu lệnh: *"nohup python3 socker_server.py &"*

Bước 4: Chạy module TransferStategy bằng câu lệnh: "*TransferStategy faust -A data_process_app worker -l info"*

Tại IoT Agent: Khởi chạy module gửi file bằng câu lệnh: *"python3 socket_client.py"*. Dữ liệu được gửi chứa trong client_data/test.

3.2. Kịch bản thử nghiệm và bộ tiêu chí đánh giá

3.2.1 Kịch bản thử nghiệm

Tại phần thử nghiệm, học viên sử dụng 250 tệp lệnh gọi hệ thống dưới dạng đồ thị (system call graph). Các tệp này được gửi từ phía IoT Agent cho IoT Analyzer, có định dạng ".adjlist" và nặng tối đa 9.9 MB.

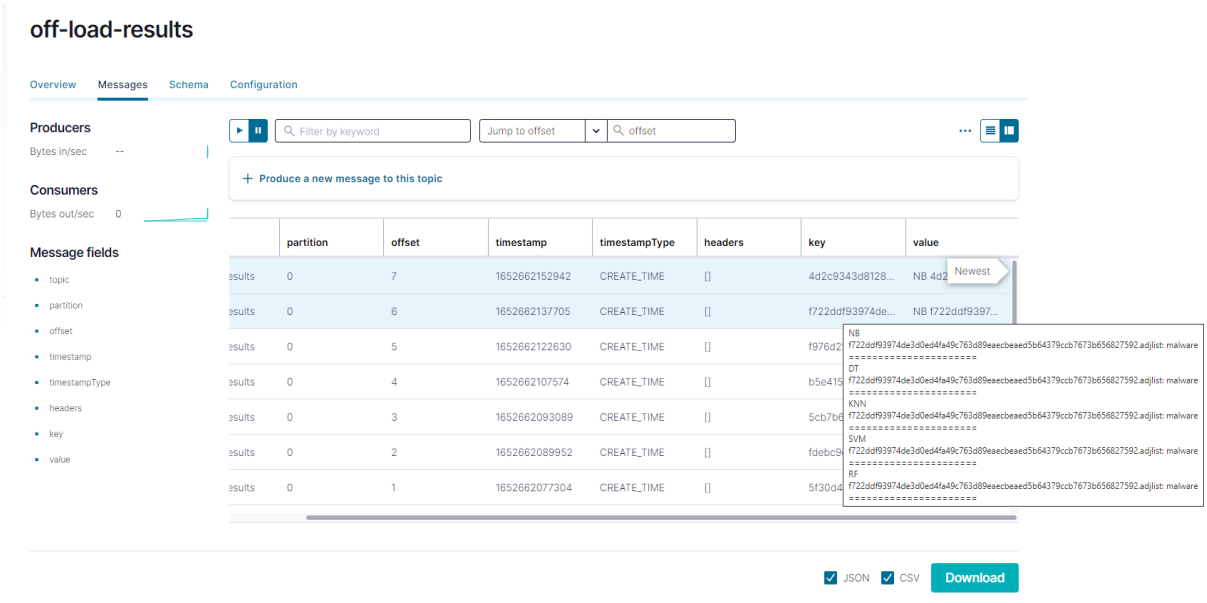
a) Kịch bản với trường hợp thử nghiệm chưa áp dụng phân tán tác vụ.

Kịch bản 1: Khi IoT Analyzer làm việc ở trạng thái không quá tải.

Bước 1: Thực hiện đẩy dữ liệu chưa được xử lý từ IoT Agent.

```
(base) root@center-server:/home/center/Master_prj/distributed-model-thesis/src/agent# python3 socket_client.
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0a1830c9206118843062d06d668c89622b88bea97fdb189e80b060d50905e7e3.adj
[RECEIVED] FILE NAME
UPLOADED: 0a1830c9206118843062d06d668c89622b88bea97fdb189e80b060d50905e7e3.adjlist
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0c3583b5cd35e822cc0d9f2fca77c3ee7fa7208e7028d3f38cc623f51e165218.adj
[RECEIVED] FILE NAME
UPLOADED: 0c3583b5cd35e822cc0d9f2fca77c3ee7fa7208e7028d3f38cc623f51e165218.adjlist
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0c8848d7211ccedc4f06af9a00086598d59186820a37cf1c91ccba00552950b1.adj
[RECEIVED] FILE NAME
UPLOADED: 0c8848d7211ccedc4f06af9a00086598d59186820a37cf1c91ccba00552950b1.adjlist
```

Bước 2. Thực hiện phân tích trên IoT Analyzer.



Bước 3: Thu thập kết quả.

Kịch bản 2: Khi IoT Analyzer quá tải CPU và RAM.

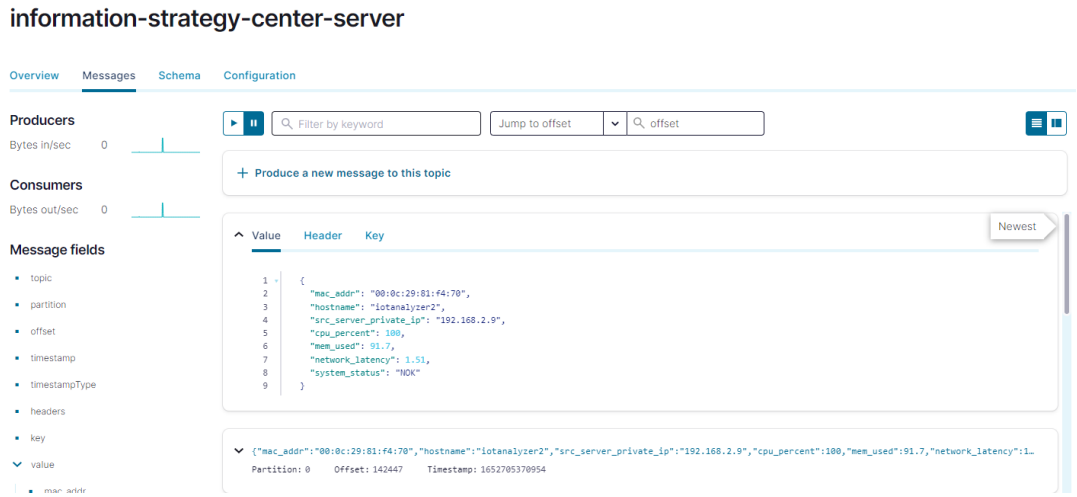
Bước 1: Tạo môi trường giả lập đầy CPU sử dụng thư viện stress.

```
apt install stress-ng
stress-ng --vm-bytes $(awk '/MemAvailable/{printf "%d\n", $2 * 0.9;}' < /proc/meminfo)k --vm-keep -m 1
```

Bước 2: Thực hiện đẩy dữ liệu chưa được xử lý từ IoT Agent.

```
(base) root@center-server:/home/center/Master_prj/distributed-model-thesis/src/agent# python3 socket_client.py
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0a1830c9206118843062d06d668c89622b88bea97fdb189e80b060d50905e7e3.adjlist
[RECEIVED] FILE NAME
UPLOADED: 0a1830c9206118843062d06d668c89622b88bea97fdb189e80b060d50905e7e3.adjlist
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0c3583b5cd35e822cc0d9f2fca77c3ee7fa7208e7028d3f38cc623f51e165218.adjlist
[RECEIVED] FILE NAME
UPLOADED: 0c3583b5cd35e822cc0d9f2fca77c3ee7fa7208e7028d3f38cc623f51e165218.adjlist
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0c8848d7211ccedc4f06af9a00086598d59186820a37cf1c91ccba00552950b1.adjlist
[RECEIVED] FILE NAME
UPLOADED: 0c8848d7211ccedc4f06af9a00086598d59186820a37cf1c91ccba00552950b1.adjlist
```

Bước 3. IoT Analyzer bị quá tải nhưng vẫn phải xử lý tác vụ.



Bước 4: Thu thập kết quả.

b) Kịch bản với trường hợp thử nghiệm áp dụng mô hình phân tán tác vụ.

Kịch bản 3: Khi IoT Analyzer quá tải CPU và RAM.

Bước 1: Tạo môi trường giả lập đầy CPU sử dụng thư viện stress.

```
apt install stress-ng
stress-ng --vm-bytes $(awk '/MemAvailable/{printf "%d\n", $2 * 0.9;}' < /proc/meminfo)k --vm-keep -m 1
```

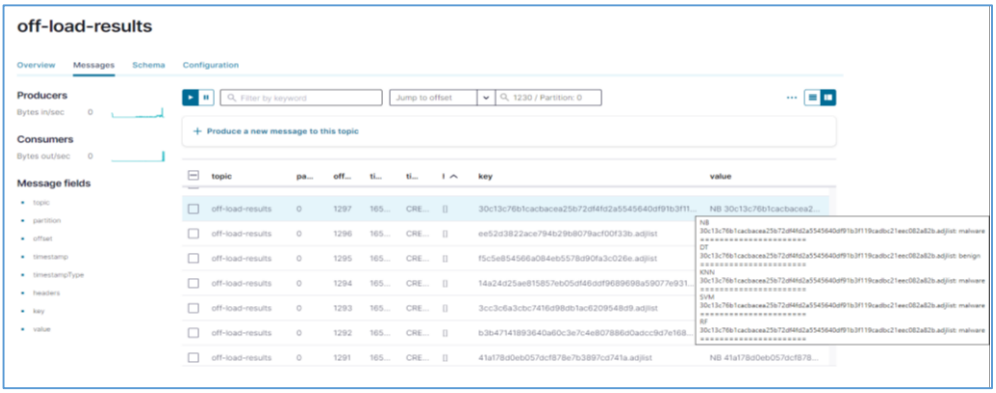
Bước 2: Thực hiện đẩy dữ liệu chưa được xử lý từ IoT Agent.

```
(base) root@center-server:/home/center/Master_prj/distributed-model-thesis/src/agent# python3 socket_client.py
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0a1830c9206118843062d06d668c89622b88bea97fdb189e80b060d50905e7e3.adjlist
[RECEIVED] FILE NAME
UPLOADED: 0a1830c9206118843062d06d668c89622b88bea97fdb189e80b060d50905e7e3.adjlist
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0c3583b5cd35e822cc0d9f2fca77c3ee7fa7208e7028d3f38cc623f51e165218.adjlist
[RECEIVED] FILE NAME
UPLOADED: 0c3583b5cd35e822cc0d9f2fca77c3ee7fa7208e7028d3f38cc623f51e165218.adjlist
OK@Welcome to the File Server.
Uploading:  ./client_data/scg/data/test/0c8848d7211ccedc4f06af9a00086598d59186820a37cf1c91ccba00552950b1.adjlist
[RECEIVED] FILE NAME
UPLOADED: 0c8848d7211ccedc4f06af9a00086598d59186820a37cf1c91ccba00552950b1.adjlist
```

Bước 3. IoT Analyzer bị quá tải sẽ phân tán tác vụ cho IoT Analyzer khác rảnh rỗi thông qua mô hình.


```
> [IPv4 ('192.168.2.10', 9092)]>: Closing connection.
2022-05-16 00:47:41 iotanalyzer2 root[81486] INFO ==================================================2[END TIMER
NE TASK] c68aa52063751ae1bf693feee78a7029.adjlist
server_data/local/fdebc9cefbdb1aa259e49572dcad717a.adjlist
498
2022-05-16 00:47:41 iotanalyzer2 root[81486] WARNING ================SYSTEM OVERLOAD AND REQUEST SUPPORT========
======
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO <BrokerConnection node_id=bootstrap-0 host=192.168.2.8:90
connecting> [IPv4 ('192.168.2.8', 9092)]>: connecting to 192.168.2.8:9092 [('192.168.2.8', 9092) IPv4]
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO Probing node bootstrap-0 broker version
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO <BrokerConnection node_id=bootstrap-0 host=192.168.2.8:90
connecting> [IPv4 ('192.168.2.8', 9092)]>: Connection complete.
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO Broker version identified as 2.5.0
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO Set configuration api_version=(2, 5, 0) to skip auto chec
rsion requests on startup
2022-05-16 00:47:41 iotanalyzer2 root[81486] INFO DONE setting up Kafka Producer
2022-05-16 00:47:41 iotanalyzer2 root[81486] INFO SEND request to target node
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO <BrokerConnection node_id=0 host=192.168.2.8:9092 <connec
> [IPv4 ('192.168.2.8', 9092)]>: connecting to 192.168.2.8:9092 [('192.168.2.8', 9092) IPv4]
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO <BrokerConnection node_id=0 host=192.168.2.8:9092 <connec
> [IPv4 ('192.168.2.8', 9092)]>: Connection complete.
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO <BrokerConnection node_id=bootstrap-0 host=192.168.2.8:90
connected> [IPv4 ('192.168.2.8', 9092)]>: Closing connection.
2022-05-16 00:47:41 iotanalyzer2 kafka.conn[81486] INFO <BrokerConnection node_id=0 host=192.168.2.8:9092 <connec
 [IPv4 ('192.168.2.8', 9092)]>: Closing connection.
2022-05-16 00:47:41 iotanalyzer2 root[81486] INFO HANDLING FILE server_data/local/8f3d86c28ec5956fc36335dc6627f7c
60cb5d14ac6080b9a7412c9b59ca1.adjlist with node: 192.168.2.8
server_data/local/5f30d4fb19dd84c568be917661e5e5f1.adjlist
496
```

Bước 4: Thu thập kết quả.



3.2.2 Tiêu chí đánh giá

Mục đích của bộ cân bằng tải có 2 tiêu chí chính đó là:

- Đảm bảo tính sẵn sàng của hệ thống.
- Đảm bảo được việc xử lý lưu lượng traffic lớn của ứng dụng mà không bị mất dịch vụ.

Với mục đích nêu trên, có rất nhiều các tiêu chí để đánh giá sự hiệu quả của bộ cân bằng tải như đo đạc các thông số về tài nguyên sử dụng (resource ultilization), độ trễ (latency), chi phí (cost), ….

Tuy nhiên, trong khuôn khổ của luận văn, để chứng minh sự hiệu quả của giải pháp cân bằng tải động có thể giải quyết được bài toán đã nêu ở mục 2.1 không, học viên đánh giá sự hiệu quả của giải pháp theo một số thông số như sau:

- Tổng thời gian xử lý một lượng công việc giống nhau (R): Giá trị của tổng thời gian phản hồi khi thực hiện các tác vụ trên IoT Analyzer.

$$R = T_1 + T_2 + \cdots + T_n$$

Trong đó: T_i là thời gian xử lý tác vụ thứ i

- Khả năng xử lý của hệ thống (S): Được tính là khối lượng dữ liệu được xử lý trong một khoảng thời gian.

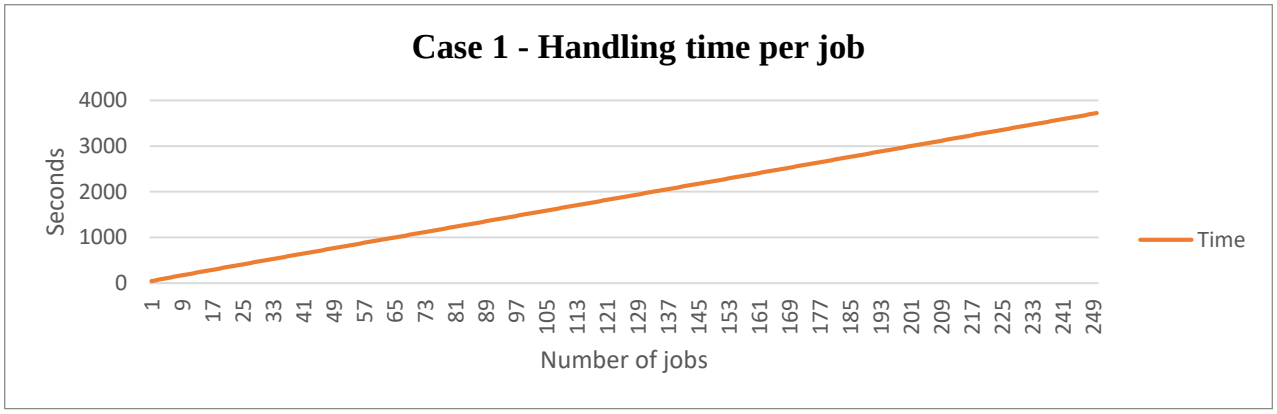
$$S = \frac{Task\ (MB)}{Time\ (s)}$$

- Hai giá trị này này sẽ được ghi lại và đánh giá qua các kịch bản khác nhau.

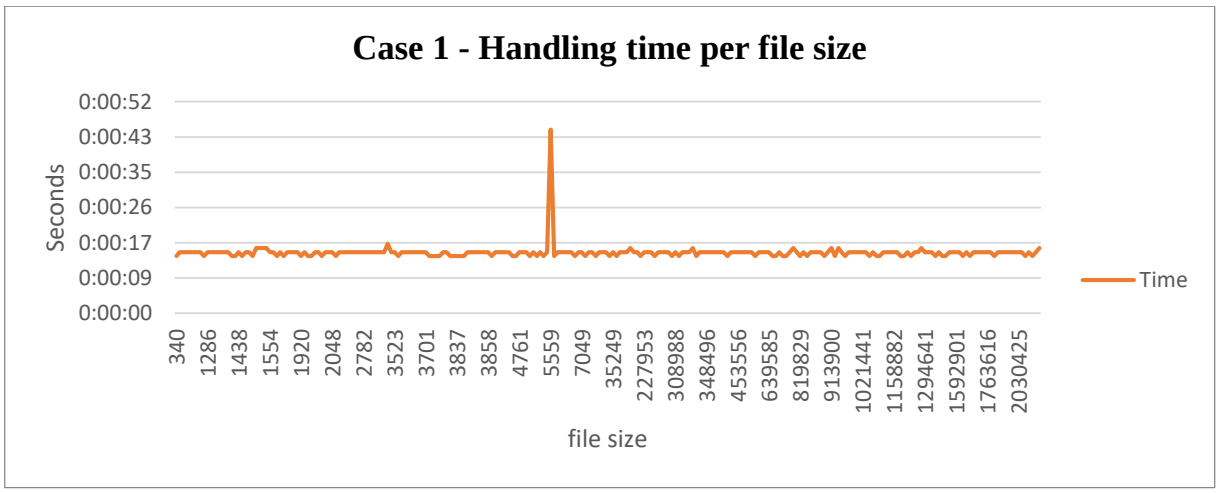
3.3. Kết quả và đánh giá

3.3.1. Kịch bản 1

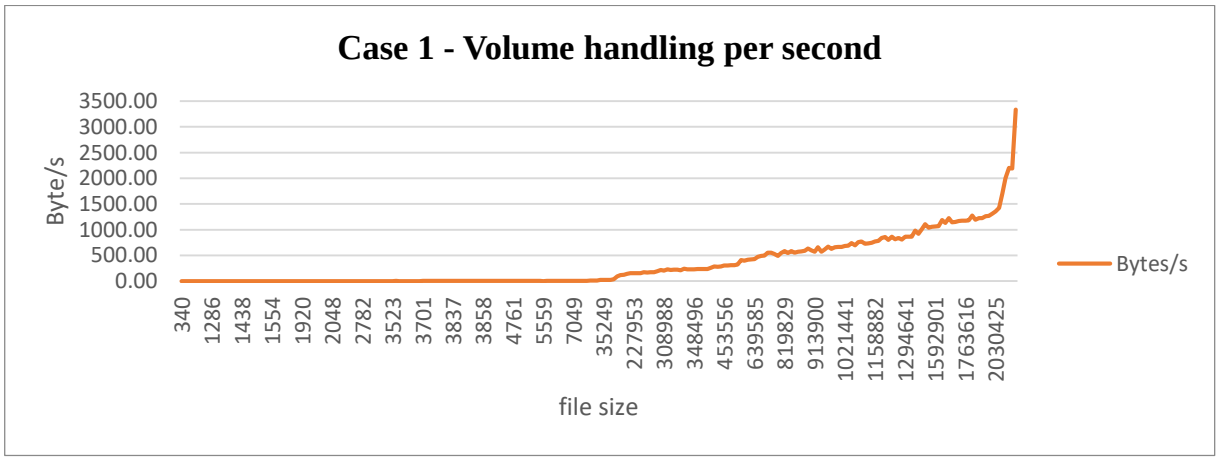
Tại kịch bản 1, kết quả cho thấy khi một nút IoT Analyzer ở trạng thái làm việc bình thường xử lý 250 tệp tin mất 3721 giây (tương đương 1 giờ 2 phút 1 giây). Thời gian xử lý trung bình một tệp tin vào khoảng 15 giây. Khối lượng xử lý của tệp tin lớn cũng gần tương đương tệp tin nhỏ cho thấy hệ thống đang hoàn toàn có thể xử lý được các tệp dữ liệu khi tài nguyên cho phép. Hình 3.2 mô tả thời gian xử lý của một IoT Analyzer với số lượng các tệp tin đầu vào. Hình 3.3 mô tả thời gian xử lý một tệp tin với kích thước khác nhau. Cuối cùng hình 3.4 cho thấy khối lượng dữ liệu hệ thống xử lý trong 1 giây khi không bị quá tải.



Hình 3.2 Kịch bản 1: Thời gian xử lý trên số lượng các tệp tin đầu vào.



Hình 3.3 Kịch bản 1: Thời gian xử lý ứng với kích cỡ của dữ liệu đầu vào.

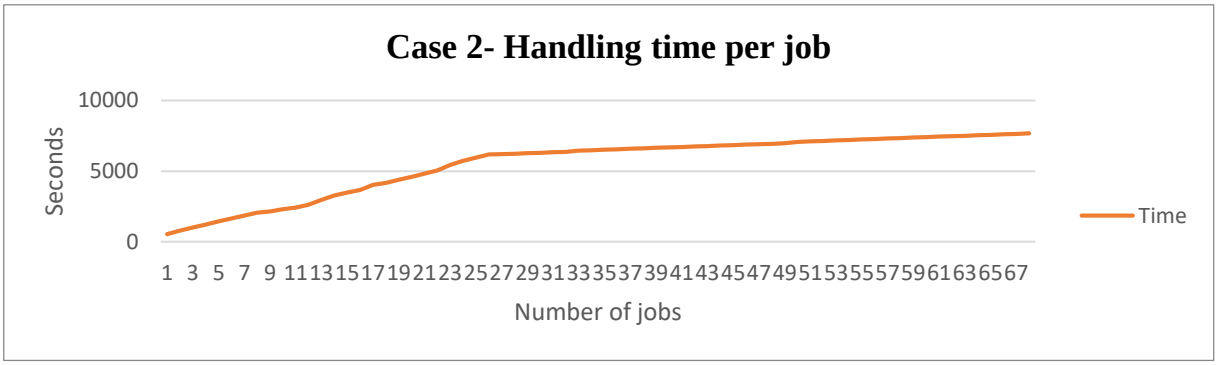


Hình 3.4 Kịch bản 1: Khối lượng xử lý ứng với kích cỡ tệp dữ liệu đầu vào.

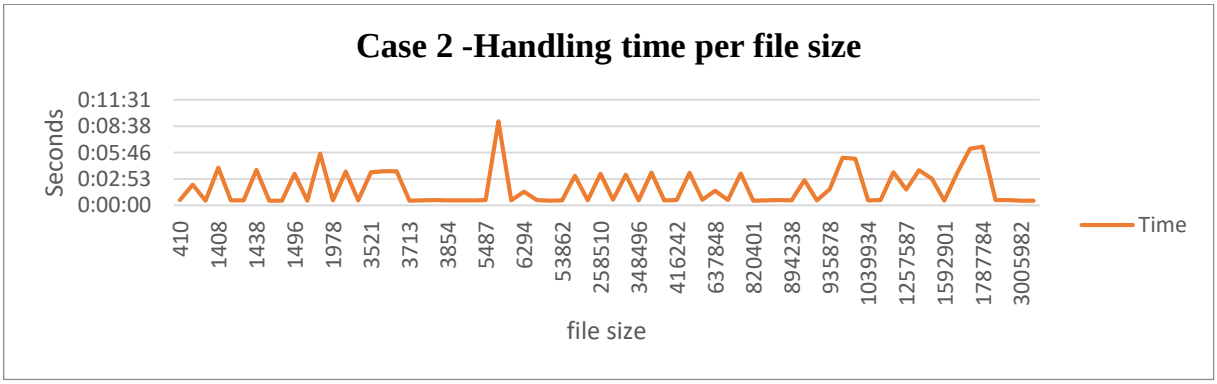
3.3.2. Kịch bản 2

Tại kịch bản 2, khi hệ thống bị quá tải cho thấy kết quả xử lý chậm đi rất đáng kể. Tải của hệ thống khi đang xử lý: CPU 100%, RAM 92%. Khi hệ thống xử lý chậm là mối tiềm tàng về bảo mật trong hệ thống. IoT Analyzer ở trạng thái quá tải xử lý 69

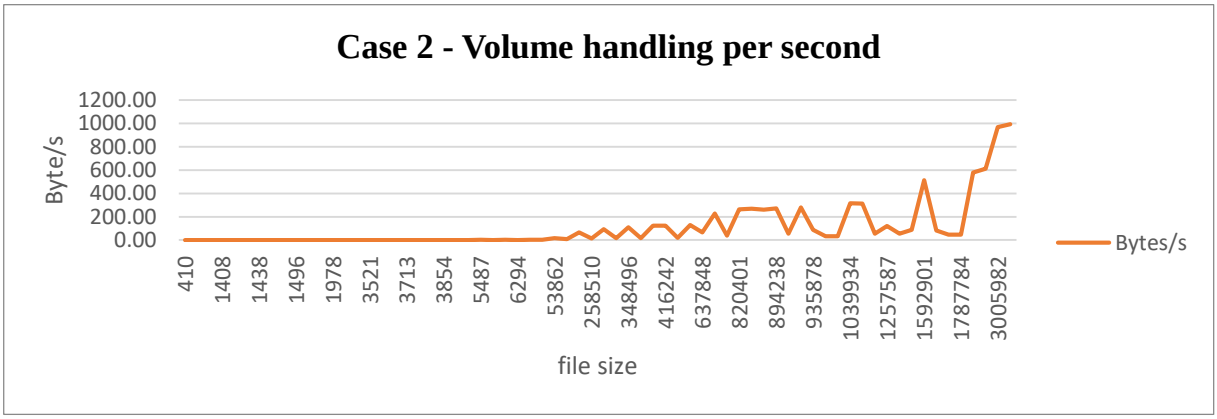
tệp tin mất 7674 giây (tương đương 2 giờ 7 phút 54 giây) gần gấp đôi so với kịch bản 1 và hiệu suất kém rất nhiều. Thời gian xử lý trung bình một tệp tin vào khoảng 1 phút 56 giây. Hình 3.5 mô tả thời gian xử lý của một IoT Analyzer với số lượng các tệp tin đầu vào. Hình 3.6 mô tả thời gian xử lý một tệp tin với kích thước khác nhau. Cuối cùng hình 3.7 cho thấy khối lượng dữ liệu hệ thống xử lý trong 1 giây khi bị quá tải.



Hình 3.5 Kịch bản 2: Thời gian xử lý trên số lượng các tệp tin đầu vào.



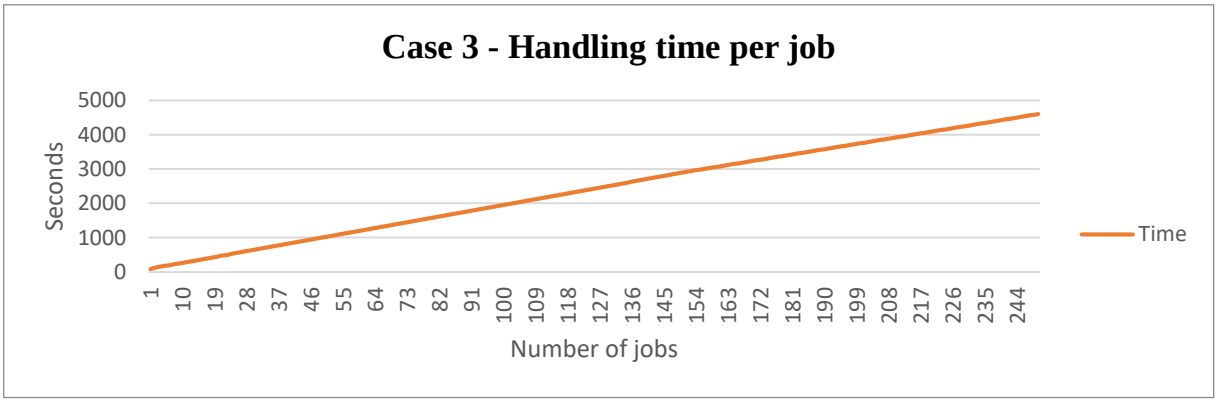
Hình 3.6 Kịch bản 2: Thời gian xử lý ứng với kích cỡ của dữ liệu đầu vào.



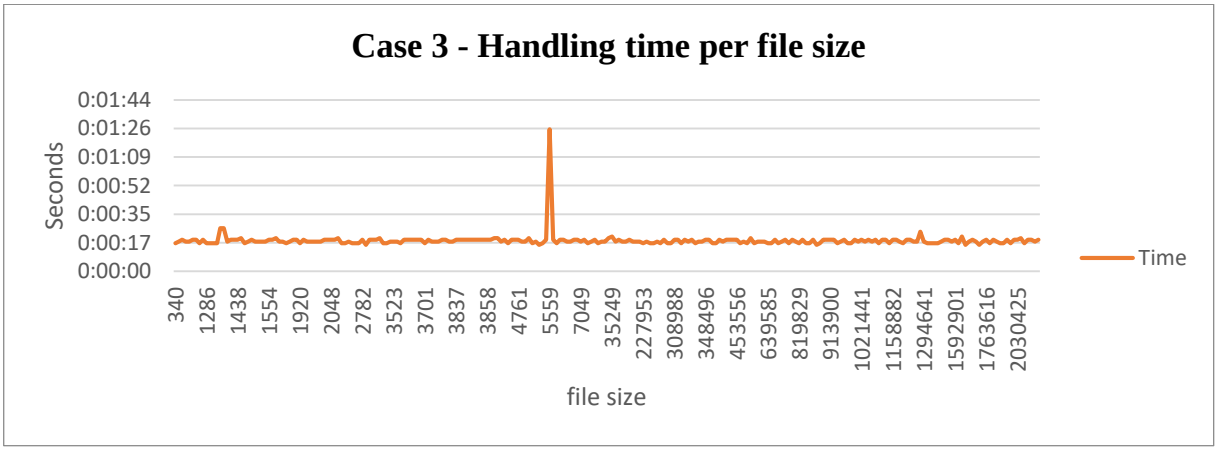
Hình 3.7 Kịch bản 2: Khối lượng xử lý ứng với kích cỡ tệp dữ liệu đầu vào.

3.3.3. Kịch bản 3

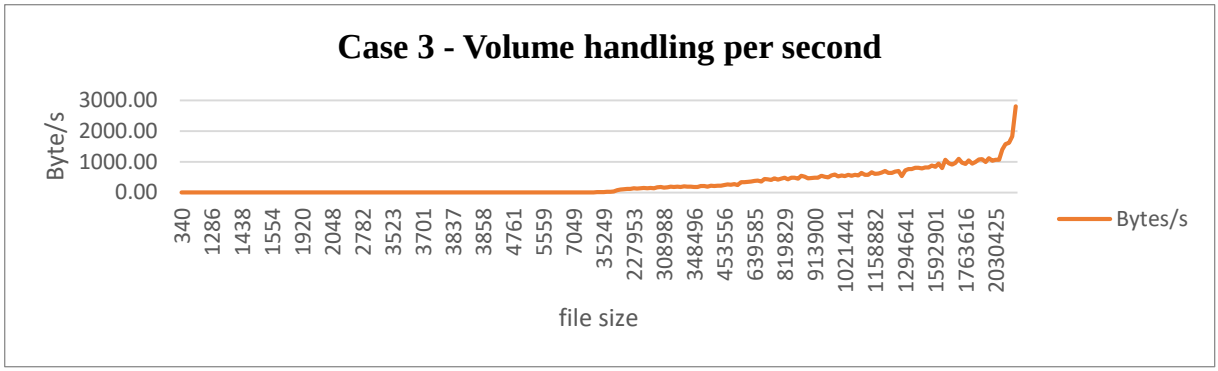
Tại kịch bản 3, khi hệ thống bị quá tải đã đẩy các tác vụ cho một nút IoT Analyzer khác. Việc này không làm gián đoạn quá trình phân tích và phát hiện mã độc IoT botnet của hệ thống, mặt khác có thể tối ưu tài nguyên rảnh rỗi. Tải của hệ thống khi đang xử lý: CPU 100%, RAM 92%, nút IoT Analyzer đích không bị quá tải có CPU, RAM < 90%. IoT Analyzer ở trạng thái quá tải xử lý 250 tệp tin mất 4608 giây (tương đương 1 giờ 16 phút 46 giây). Thời gian xử lý trung bình một tệp tin vào khoảng 18 giây. Hình 3.8 mô tả thời gian xử lý của một IoT Analyzer với số lượng các tệp tin đầu vào. Hình 3.9 mô tả thời gian xử lý một tệp tin với kích thước khác nhau. Hình 3.10 cho thấy khối lượng dữ liệu hệ thống xử lý trong 1 giây khi bị quá tải. Cuối cùng Hình 3.11 là biểu đồ so sánh giữa các kịch bản, cho thấy hiệu quả của việc phân tán tác vụ trong hệ thống.



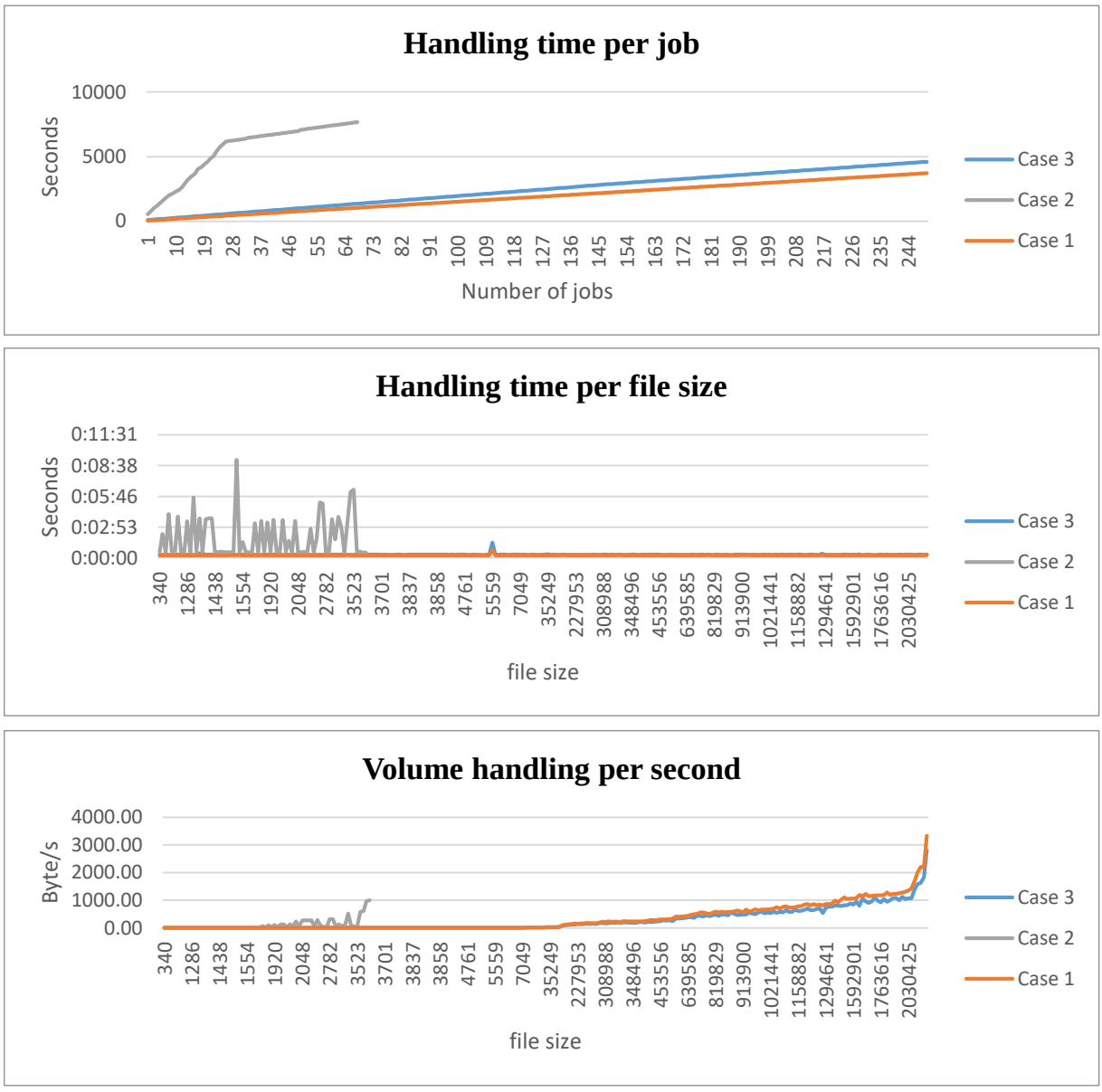
Hình 3.8 Kịch bản 3: Thời gian xử lý trên số lượng các tệp tin đầu vào.



Hình 3.9 Kịch bản 3: Thời gian xử lý ứng với kích cỡ của dữ liệu đầu vào.



Hình 3.10 Kịch bản 3: Khối lượng xử lý ứng với kích cỡ tệp dữ liệu đầu vào.



Hình 3.11 So sánh hiệu quả giữa khi áp dụng giải pháp phân tán động (kịch bản 2) và khi không áp dụng giải pháp phân tán động.

Bảng 3.2 So sánh kết quả giữa áp dụng giải pháp phân tán động (kịch bản 2) và khi không áp dụng giải pháp phân tán động.

Kịch bản	Số lượng tệp tin xử lý	Tổng thời gian xử lý	Trung bình thời gian xử lý	Số byte xử lý trung bình trong 1 giây
1	250	1:02:01	0:15	33573.13
2	69	2:07:54	1:56	10940.48
3	250	1:16:48	0:19	27383.72

Kết luận chương 3.

Trong chương này, học viên đã đưa ra môi trường thử nghiệm sát với thực tế nhất đối với mô hình đề xuất. Ngoài ra, học viên đã đưa ra một số kịch bản thực nghiệm trong trường hợp không có cơ chế cân bằng tải và có có chế cân bằng tải. Đồng thời đưa ra các tham số để đánh giá khả năng phân tán tác vụ của hệ thống. Kết quả đạt được khi thử nghiệm tương đối khả quan khi áp dụng cơ chế phân tán tác vụ nhằm tăng cường khả năng phát hiện mã độc một cách nhanh chóng. Điều này cũng cho thấy tiềm năng của mô hình và có thể được tối ưu, sửa đổi và phát triển trong tương lai.

KẾT LUẬN

Qua thời gian nghiên cứu, tìm hiểu, tiếp cận các tri thức về lĩnh vực bảo mật trong IoT, đặc biệt là phát hiện mã độc IoT Botnet, luận văn đã đưa ra mô hình phát hiện mã độc IoT Botnet, được công bố tại hội nghị Computer Science On-line Conference thứ 11 năm 2022. Đặc biệt, để nâng cao khả năng của hệ thống khi thiết bị IoT Analyzer bị quá tải, luận văn đã đưa ra mô hình phân tán động nhằm phân tán tác vụ và thực hiện phân tích mã độc một cách nhanh chóng nhất. Luận văn "Nghiên cứu xây dựng giải pháp phân tán động trong hệ thống phân tích mã độc IoT botnet dựa trên kafka và ksql" đã được học viên hoàn thành và đạt được các mục tiêu nghiên cứu. Các vấn đề được trình bày bao gồm:

- Tổng quan cơ sở lý thuyết mã độc trong IoT, mã độc IoT botnet, từ đó đưa ra một mô hình phân tích và phát hiện mã độc IoT Botnet.
- Tổng quan về giải pháp phân tán nói chung, đặc biệt là hệ thống phân tán động được áp dụng rất nhiều trong việc tránh quá tải ngày nay.
- Trình bày về bài toán cần xử lý trong mô hình phân tích và phát hiện mã độc IoT botnet và đưa ra một mô hình giải pháp phân tán động tác vụ dựa trên các công nghệ mới nhất hiện nay là Kafka và KSQL.
- Cuối cùng, luận văn trình bày môi trường thử nghiệm hệ thống, xây dựng chi tiết các kịch bản đánh giá cũng như bộ tham số đánh giá.

Qua kết quả thử nghiệm cho thấy, hiệu năng của hệ thống phân tích và phát hiện mã độc được đảm bảo khi sử dụng cơ chế phân tán tác vụ động. Từ đó có thể nâng cao độ tin cậy và khả năng phân tích, phát hiện mã độc IoT botnet trên các thiết bị IoT. Thử nghiệm cho kết quả tương đối khả quan và cho thấy tiềm năng ứng dụng của mô hình trong tương lai. Tuy nhiên, mô hình trong luận văn cũng cần cải thiện khi dữ liệu IoT mở rộng ra rất lớn và mô hình học máy cần được cập nhật. Trong tương lai, học viên sẽ tiếp tục nghiên cứu và tối ưu mô hình bằng cách áp dụng các giải pháp xử lý dữ liệu lớn cũng như mô hình học máy hiện đại để đem lại những hiệu quả tốt hơn.

Trên đây là một số kết luận và hướng nghiên cứu của luận văn. Học viên xin chân thành cảm ơn các quý thầy, cô đã hướng dẫn khoa học và phản biện để giúp học viên hoàn thiện những thiếu sót trong luận văn cũng như định hướng cho việc phát triển đề tài trong tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] K. Ashton, ‘That “Internet of Things” Thing’, p. 1.

[8] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, ‘DDoS in the IoT: Mirai and other botnets’, *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[9] P. Beltrán-García, E. Aguirre-Anaya, P. J. Escamilla-Ambrosio, and R. Acosta-Bermejo, ‘IoT Botnets’, *Telematics and Computing - 8th International Congress, WITCOM 2019, Proceedings*, pp. 247–257, Jan. 2019, doi: 10.1007/978-3-030-33229-7_21.

[10] H. Alzahrani, M. Abulkhair, and E. Alkayal, ‘A Multi-Class Neural Network Model for Rapid Detection of IoT Botnet Attacks’, *International Journal of Advanced Computer Science and Applications*, vol. 11, Jan. 2020, doi: 10.14569/IJACSA.2020.0110783.

[11] M. Wazzan, D. Algazzawi, O. Bamasaq, A. Albeshri, and L. Cheng, ‘Internet of Things Botnet Detection Approaches: Analysis and Recommendations for Future Research’, *Applied Sciences*, vol. 11, no. 12, Art. no. 12, Jan. 2021, doi: 10.3390/app11125713.

[12] S. Hilt, ‘Worm War: The Botnet Battle for IoT Territory’, p. 30.

[13] Q.-D. Ngo, H.-T. Nguyen, H.-A. Tran, and D.-H. Nguyen, ‘IoT Botnet detection based on the integration of static and dynamic vector features’, in *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*, 2021, pp. 540–545.

[14] ‘A study of IoT malware activities using association rule learning for darknet sensor data | International Journal of Information Security’. https://dl.acm.org/doi/abs/10.1007/s10207-019-00439-w (accessed Apr. 25, 2022).

[15] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, ‘graph2vec: Learning distributed representations of graphs’, *arXiv preprint arXiv:1707.05005*, 2017.

[16] H.-V. Le and Q.-D. Ngo, ‘V-sandbox for dynamic analysis IoT botnet’, *IEEE Access*, vol. 8, pp. 145768–145786, 2020.

[17] J. D. G. Coulouris, ‘Distributed Systems: Concepts and Design’, *DISTRIBUTED SYSTEMS*, p. 1067.

[18] A. S. Tanenbaum and M. van Steen, *Distributed systems: principles and paradigms*, 2nd ed. Upper Saddle RIiver, NJ: Pearson Prentice Hall, 2007.

[19] M. Randles, D. Lamb, and A. Taleb-Bendiab, ‘A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing’, in *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, USA, Apr. 2010, pp. 551–556. doi: 10.1109/WAINA.2010.85.

[20] ‘Biased random walks on resource network graphs for load balancing | The Journal of Supercomputing’. https://dl.acm.org/doi/10.1007/s11227-009-0366-6 (accessed Jul. 16, 2021).

[21] M. Randles, A. Taleb-Bendiab, and D. Lamb, ‘Cross Layer Dynamics in Self-Organising Service Oriented Architectures’, in *Self-Organizing Systems*, Berlin, Heidelberg, 2008, pp. 293–298. doi: 10.1007/978-3-540-92157-8_28.

[22] M. Randles, A. Taleb-Bendiab, and D. Lamb, ‘Scalable Self-Governance Using Service Communities as Ambients’, in *2009 Congress on Services - I*, Jul. 2009, pp. 813–820. doi: 10.1109/SERVICES-I.2009.93.

[23] ‘On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers - Sunil Nakrani, Craig Tovey, 2004’. https://journals.sagepub.com/doi/abs/10.1177/105971230401200308 (accessed Jul. 16, 2021).

[24] M. Dorigo, V. Maniezzo, and A. Colorni, ‘Ant system: optimization by a colony of cooperating agents’, *IEEE Transactions on Systems, Man, and Cybernetics*,

Part B (Cybernetics), vol. 26, no. 1, pp. 29–41, Feb. 1996, doi: 10.1109/3477.484436.

[25] 'Ant colony optimization | IEEE Journals & Magazine | IEEE Xplore'. https://ieeexplore.ieee.org/abstract/document/4129846/ (accessed Jul. 16, 2021).

[26] 'A new pheromone updating strategy in ant colony optimization | IEEE Conference Publication | IEEE Xplore'. https://ieeexplore.ieee.org/abstract/document/1380766 (accessed Jul. 16, 2021).

[27] C.-W. Chiang, Y.-C. Lee, C.-N. Lee, and T.-Y. Chou, 'Ant colony optimisation for task matching and scheduling', *IEE Proceedings - Computers and Digital Techniques*, vol. 153, no. 6, pp. 373–380, Nov. 2006, doi: 10.1049/ip-cdt:20050196.

Website

[2] 'Y.2060 : Overview of the Internet of things'. https://www.itu.int/rec/T-REC-Y.2060-201206-I (accessed Apr. 16, 2022).

[3] 'Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper', *Cisco*. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed Feb. 23, 2022).

[4] '2020 Unit 42 IoT Threat Report 2020 Unit 42 IoT Threat Report', *Unit42*, Mar. 10, 2020. https://unit42.paloaltonetworks.com/iot-threat-report-2020/ (accessed Apr. 20, 2022).

[5] 'What is Malware? - Definition and Examples', *Cisco*. https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html (accessed Apr. 21, 2022).

[6] ‘IoT Botnet - Definition’. https://www.trendmicro.com/vinfo/us/security/definition/iot-botnet (accessed Apr. 21, 2022).

[7] ‘IoT role in Dyn cyberattack | Kaspersky official blog’. https://www.kaspersky.com/blog/attack-on-dyn-explained/13325/ (accessed Feb. 23, 2022).

[28] ‘Apache Kafka’, *Apache Kafka*. https://kafka.apache.org/ (accessed Dec. 18, 2020).

[29] ‘Quick Start for Confluent Platform | Confluent Documentation’. https://docs.confluent.io/platform/current/quickstart/ce-docker-quickstart.html (accessed May 11, 2022).

PHỤ LỤC

Mã nguồn chương trình chính:

```
import os
import shutil
import socket
import threading
import queue
import time

from psutil import cpu_percent, virtual_memory
import logging
import coloredlogs
from TransferStategy import TransferStrategy
from LocationStategy import LocationStrategy
from InformationStategy import InformationStrategy
from dotenv import load_dotenv
from concurrent.futures import ThreadPoolExecutor

coloredlogs.install()
load_dotenv(dotenv_path=".env")
'''
This module have responsibility:
1. Receive raw data from IoT Agent
2. Send result to dashboard
'''

OFFLOAD_TASKS_TOPIC = os.environ.get("TOPIC_OFF_LOAD_TASKS")
OFFLOAD_RESULT_TOPIC = os.environ.get("TOPIC_OFF_LOAD_result")
KAFKA_PORT = os.environ.get("KAFKA_PORT")
DASHBOARD = os.environ.get("DASHBOARD")
DASHBOARD_PORT = os.environ.get("DASHBOARD_PORT")

transfer = TransferStrategy()
location = LocationStrategy()
location.create_ksql_stream()
info = InformationStrategy()

IP = "192.168.2.9"
PORT = 4456
ADDR = (IP, PORT)
```

```
size = 4096
format_encode = "utf-8"
server_data_path = "server_data/local"
print("[STARTING] Server is starting")
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)
server.listen()
print(f"[LISTENING] Server is listening on {IP}:{PORT}.")
q_tasks = queue.Queue()


def active_socket_server():
    location.create_ksql_stream()
    #executor = ThreadPoolExecutor(1)
    while True:
        conn, addr = server.accept()
        #thread = threading.Thread(target=handle_client, args=(conn, addr))
        #thread.start()
        print(f"[ACTIVE CONNECTIONS] {threading.active_count() - 1}")
        conn.send("OK@Welcome to the File Server.".encode(format_encode))
        name = conn.recv(size).decode(format_encode)
        print(f"[RECEIVING] {name}")

        conn.send("[RECEIVED] FILE NAME".encode(format_encode))
        name = name.split("/")[-1]
        filepath = os.path.join(server_data_path, name)
        #os.system("rm -rf ./server_data/local/*")
        with open(filepath, "w") as f:
            while True:
                text = conn.recv(size).decode(format_encode)
                # print(text)
                f.write(text)
                if not text:
                    break
        f.close()
        # self.task_state = True
        ##add filename into queue
        q_tasks.put(filepath)
        logging.info(filepath)
        print(q_tasks.qsize())
        #thread = threading.Thread(target=handle_data(filepath)) #, args=(f"{filepath}"))
```

```
        #thread.start()
        #executor.submit(handle_data, (filepath))
        print(f"[DISCONNECTED] {addr} disconnected")
        conn.close()

logging.warning(f"=========================================1[STAR
TING TIMER][TASK] {name}\n")



        #if q_tasks.qsize() != 0:
        #    handle_data(q_tasks.get())


def handle_client(conn, addr):
    while True:
        print(f"[NEW CONNECTION] {addr} connected.")
        conn.send("OK@Welcome to the File Server.".encode(format_encode))
        name = conn.recv(size).decode(format_encode)
        print(f"[RECEIVING] {name}")

        conn.send("[RECEIVED] FILE NAME".encode(format_encode))
        name = name.split("/")[-1]
        filepath = os.path.join(server_data_path, name)
        #os.system("rm -rf ./server_data/local/*")
        with open(filepath, "w") as f:
            while True:
                text = conn.recv(size).decode(format_encode)
                # print(text)
                f.write(text)
                if not text:
                    break
        f.close()
        # self.task_state = True
        ##add filename into queue
        q_tasks.put(filepath)
        print(f"[DISCONNECTED] {addr} disconnected")
        conn.close()
        #print(f"[TASK] {name}\n")


def handle_data(filepath):
```

```
    with open(filepath, "r") as f:
        data = f.read()
    f.close()
    src_ip = info.get_private_ip()
    name = filepath.split("/")[-1]
    key_info = src_ip+":"+name
    ##Check system state overload or not
    if cpu_percent() > 101.0 and virtual_memory()[2] > 90.0:
        logging.warning("=================SYSTEM OVERLOAD AND
REQUEST SUPPORT==================")
        #location.create_ksql_stream()
        #location.query_candidates_with_low_latency()
        choosen_node = location.select_the_best()[1][0]
        transfer.kafka_publish_data(kafka_broker_ip=choosen_node,
                                    kafka_port=KAFKA_PORT,
                                    data_key=key_info,
                                    data_value=data)
        logging.info("HANDLING FILE %s with node: %s" % (q_tasks.get(),
choosen_node))
    else:
        logging.info("=============SYSTEM UNDERLOAD AND HANDLE
REQUEST BY ITS SELF============")
        os.system("rm -rf ./server_data/test_dataset/test_tasks/*")
        shutil.copyfile(filepath, f"./server_data/test_dataset/test_tasks/{name}")
        os.system("python3.8 ./Graph2vec.py")
        os.system("python3.8 ./defensive_model.py")
        with open("./result/pred_result.txt", "r") as f:
            pred_result = f.read()
        f.close()
        transfer.kafka_publish_data(kafka_broker_ip="192.168.2.9",
topic=OFFLOAD_RESULT_TOPIC, data_key=name, data_value=pred_result)
        transfer.kafka_publish_data(kafka_broker_ip="192.168.2.10",
topic=OFFLOAD_RESULT_TOPIC, data_key=name, data_value=pred_result)

logging.info("================================================2
[END TIMER][DONE TASK] %s" % name)



def while_():
    while True:
```

```
        #logging.info(f"==============={q_tasks.qsize()}=============")
        if q_tasks.qsize() > 0:
            file_path = q_tasks.get()
            print(file_path)
            print(q_tasks.qsize())
            handle_data(file_path)



def main():
    executor = ThreadPoolExecutor(1)
    executor.submit(while_)
    active_socket_server()


if __name__ == "__main__":
    main()
```