ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG

Nguyễn Vũ Minh Thành

THỰC HIỆN NỀN TẢNG IOT SỬ DỤNG HỆ THỐNG BACK-END GIAO TIẾP PHẦN CỨNG QUA WIFI KẾT HỢP KỸ THUẬT MẬT MÃ HÓA NHẸ

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHƯƠNG TRÌNH CHẤT LƯỢNG CAO

Tp. Hồ Chí Minh, tháng 07/2023

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG

Nguyễn Vũ Minh Thành - 19207103

THỰC HIỆN NỀN TẢNG IOT SỬ DỤNG HỆ THỐNG BACK-END GIAO TIẾP PHẦN CỨNG QUA WIFI KẾT HỢP KỸ THUẬT MẬT MÃ HÓA NHẸ

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN
NGÀNH KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG
CHƯƠNG TRÌNH CHẤT LƯỢNG CAO

NGƯỜI HƯỚNG DẪN KHOA HỌC
TS. Lê Đức Hùng

Tp. Hồ Chí Minh, tháng 07/2023

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn sâu sắc tới TS. Lê Đức Hùng. Trong thời gian qua, thầy đã giúp đỡ và chỉ bảo tận tình cho em để hoàn thành khóa luận tốt nghiệp này. Hơn thế nữa, thầy đã tạo cơ hội và điều kiện để em được tham gia vào các dự án và báo khoa học. Chính vì vậy, em được tiếp cận và học hỏi nhiều về việc nghiên cứu và dự án khoa học. Và sau cùng, em đã ứng dụng được những kiến thức khoa học bổ ích từ thầy để hoàn thành khóa luận tốt nghiệp của mình.

Tiếp theo, em xin gửi lời cảm ơn chân thành tới bạn Nguyễn Văn Nhị. Nhị đã giúp đỡ em tìm hiểu và học hỏi nhiều vấn đề liên quan đến lập trình firmware. Ngoài ra, Nhị đã giúp em thiết kế bo mạch phần cứng phục vụ cho khóa luận này. Và cuối cùng, có thể nói rằng Nhị đã giúp em rất nhiều về thiết kế và kỹ thuật phần cứng.

Lời tiếp theo, em xin cảm ơn anh Hồ Như Tuấn. Anh Tuấn đã tạo nhiều điều kiện giúp em sử dụng bo mạch để thử nghiệm tại các trạm BTS.

Tiếp theo nữa, em xin cảm ơn bạn Trần Tấn Tài. Tài đã giúp em rất nhiều trong việc thiết lập bo phần cứng.

Hơn nữa, em xin chân chân thành ơn tới các thành viên trong phòng thí nghiệm DESLab. DESLab đã cho em môi trường lý tưởng để làm việc và phát triển. Hơn nữa, DESLab giúp em tiếp cận những máy móc và thiết bị khoa học tiên tiến để em làm việc, học tập, và làm khóa luận.

Sau cùng, em xin cảm ơn tới quý thầy, cô trong khoa Điện tử - Viễn Thông. Các thầy, cô đã giúp đỡ em rất nhiều về học tập và nghiên cứu.

Thành phố Hồ Chí Minh, ngày 16 tháng 8 năm 2023
Sinh viên thực hiện

Nguyễn Vũ Minh Thành

ABSTRACT

To assist managers of the Vietnam Posts and Telecommunications Group (VNPT) Tien Giang in monitoring Base Transceiver Stations (BTSs), an Internet of things (IoT) system and model were required to be constructed. The system must develop communication and synchronization techniques between sensors, physical devices, and a database in a Virtual Private Server (VPS), as well as a customized graphical user interface for data visualization. To achieve the above objectives, the IoT system will be constructed using a Microcontroller Unit (MCU) communicating things; server applications managing data; a customizable graphical interface; communication and synchronization mechanisms; and lightweight cryptography. The IoT system uses STM32 MCU communicating things, ESP32 IoT gateway, self-designed protocol, and lightweight cryptography ChaCha20-Poly1305. The system runs server applications on the VPS, including an API server, a Message Queueing Telemetry Transport (MQTT) broker, and a MongoDB database, as well as a web server with a graphical interface. In the sensing system, STM32 MCU has communication and synchronization mechanisms with the back-end system, and the ESP32 gateway has a data-routing mechanism. Server applications are built in the VPS via Docker Containerization, an Nginx web server administers and provides a web interface, and cryptography is applied during the gateway-API server transfer. The suggested IoT system has a model that is suited for communication and synchronization methods between devices, gateways, and the back-end system; the web interface is versatile in terms of creating and displaying widgets. The lightweight cryptographic algorithm assures data security between the gateway and the back-end system, and the protocol includes a dependable structure for frame detection and error detection.

TÓM TẮT

Một hệ thống và mô hình Internet of things (IoT) được yêu cầu xây dựng để giúp các quản trị viên Vietnam Posts and Telecommunications Group (VNPT) Tiền Giang giám sát các Base Transceiver Station (BTS). Hệ thống phải tạo ra cơ chế giao tiếp và đồng bộ giữa các cảm biến và thiết bị vật lý với cơ sở dữ liệu trên Virtual Private Server (VPS) và giao diện người dùng linh hoạt để trực quan hóa dữ liệu. Hệ thống IoT sẽ được xây dựng để có bo mạch Microcontroller Unit (MCU) giao tiếp vạn vật; các ứng dụng server xử lý dữ liệu; giao diện linh hoạt; cơ chế giao tiếp và đồng bộ; và mật mã hóa nhẹ, phục vụ cho các yêu cầu trên. Hệ thống sử dụng MCU STM32 giao tiếp vạn vật, IoT gateway ESP32, giao thức tự thiết kế, và mật mã hóa nhẹ ChaCha20-Poly1305. Trên VPS, hệ thống vận hành các ứng dụng server: Application programming interface (API) server, Message Queueing Telemetry Transport (MQTT) Broker và cơ sở dữ liệu MongoDB; và máy chủ web cung cấp giao diện người dùng. Trên hệ thống vạn vật, MCU STM32 đã có cơ chế giao tiếp và đồng bộ dữ liệu với back-end system, và gateway ESP32 có cơ chế chuyển tiếp dữ liệu. Trên VPS, các ứng dụng server được triển khai bằng kỹ thuật Docker Containerizing, máy chủ web Nginx quản lý và cung cấp giao diện web, và mã hóa nhẹ được sử dụng giữa giao tiếp gateway-API server. Hệ thống IoT đưa ra có mô hình phù hợp với cơ chế giao tiếp và đồng bộ giữa thiết bị, gateway, và back-end system, giao diện web có sự linh hoạt trong việc tạo và hiển thị các widget. Về việc giao tiếp, thuật toán mã hóa nhẹ đảm bảo việc bảo mật dữ liệu giữa gateway và back-end system và giao thức có cấu trúc linh hoạt trong việc nhận diện và phát hiện lỗi.

MỤC LỤC

DANH SÁCH CÁC HÌNH

DANH SÁCH CÁC BẢNG

CHƯƠNG 1: GIỚI THIỆU

Chương 1 đưa ra các nội dung tổng quan của khóa luận tốt nghiệp. Trong đó, các nội dung được trình bày là việc đặt vấn đề, mục tiêu hướng tới, giải pháp đề xuất, đóng góp, và bố cục của khóa luận.

1.1 Đặt vấn đề

Các trạm viễn thông Vietnam Posts and Telecommunications Group (VNPT) Tiền Giang yêu cầu xây dựng hệ thống Internet of things (IoT) giám sát các Base Transceiver Station (BTS). Hệ thống IoT phải cung cấp bo mạch giám sát thông số của các trạm BTS, giao diện người dùng dựa trên cơ sở dữ liệu của Virtual Private Server (VPS), và có sử dụng mật mã hóa.

Trong quá trình phát triển, mô hình hệ thống phần cứng và các cơ chế giao tiếp và đồng bộ trên phần cứng phải được thiết kế. Tại VPS, mô hình hệ thống back-end và nền tảng giao diện người dùng phải được xác định. Bên cạnh đó, hệ thống phải đưa ra loại mật mã hóa được sử dụng. Trong quá trình hoạt động, bo mạch in phải được thiết kế để phù hợp với việc giám thông số của trạm BTS và có thể triển khai cơ chế đồng bộ với back-end system. Tại VPS, kỹ thuật triển khai hệ thống back-end và nền tảng giao diện phải được đưa ra. Cuối cùng, hệ thống IoT phải đưa ra cách thức triển khai kỹ thuật mật mã hóa.

Khi yêu cầu xây dựng hệ thống IoT, các quản trị viên muốn giám sát trạm BTS online thay vì giám sát trực tiếp. Chính vì vậy, hệ thống IoT sẽ phục vụ cho việc giảm thiểu việc giám sát trạm BTS offline và chi phí di chuyển tới trạm. Không chỉ phục vụ cho việc giám sát trạm BTS, hệ thống IoT sẽ được xây dựng với cấu trúc linh hoạt để phục vụ nhiều loại hệ thống giám sát và điều khiển online.

Trong việc giao tiếp, hệ thống phải có cơ chế giao tiếp và đồng bộ đáng tin cậy và có khả năng tự phục hồi. Hơn nữa, giao diện người dùng phải linh hoạt để dễ dàng thay đổi các thành phần hiển thị của người dùng. Sau cùng, hệ thống back-end phải có cơ chế đồng bộ dữ liệu với giao diện và bo mạch một cách đáng tin cậy. Khi giải quyết được các vấn đề trên, cơ chế giao tiếp và đồng bộ có thể triển khai trên toàn hệ thống và có thể triển khai mật mã hóa hợp lý.

1.2 Mục tiêu

Xây dựng hệ thống IoT bao gồm bo mạch giám sát các BTS, hệ thống back-end thu nhận và xử lý dữ liệu phần cứng trên VPS, và giao diện cung cấp các tương tác hiển thị, giám sát và điều khiển các BTS kết hợp với các cơ chế giao tiếp và đồng bộ có bảo mật.

Cụ thể, bo mạch hướng tới thiết kế sử dụng Microcontroller Unit (MCU) cấu tạo nên các thiết bị và gateway phù hợp với việc giám sát BTS. Về việc giao tiếp, cơ chế đồng bộ được xây dựng đáng tin cậy và có khả năng tự phục hồi. Hơn nữa, cấu trúc frame giao tiếp được tạo nên có các thành phần nhận diện frame và phát hiện lỗi. Về bảo mật, mật mã hóa được triển khai hợp lý trên cơ sở giao tiếp của hệ thống. Về giao diện, hệ thống cung cấp giao diện web linh hoạt có thể tùy chỉnh các thành phần hiển thị. Về hệ thống back-end, back-end cung cấp cơ sở dữ liệu, API server, broker server, và web-server.

1.3 Giải pháp

Các IoT cloud như Blynk và ThingsBoard được sử dụng để thu nhận, quản lý và trực quan hóa dữ liệu từ bo mạch. Ngoài ra, ESP32 web-server ở chế độ station là giải pháp giám sát và điều khiển thiết bị trong phạm vi nhỏ (xung quanh một Access Point).

Việc sử dụng Blynk cloud không thể triển khai kỹ thuật mật mã hóa tùy chỉnh do nền tảng không có open-source back-end. Đối với ThingsBoard platform, nền tảng ThingsBoard cung cấp open-source cho hệ thống back-end nhưng kích thước back-end này lơn hơn khả năng tải của VPS của các quản trị viên VNPT. Trên ESP32, MCU ESP32 không đủ khả năng host một back-end phục vụ nhiều thiết bị ở các BTS.

Mô hình của hệ thống được thiết kế theo mô hình IoT 4 lớp. Theo mô hình này, hệ thống IoT bao gồm 4 lớp chính: Perception Layer, Network Layer, Processing Layer, và Application Layer. Mô hình IoT được triển khai ở các Access Point và VPS, và sử dụng các kết nối có dây và không dây.

Cụ thể, tại perception layer, phần cứng của hệ thống là bo mạch của Phòng thí nghiệm (PTN) DESLab và sử dụng MCU ESP32 và STM32 và các cảm biến đo nhiệt độ, dòng điện, điện thế, và contactor. Trong đó, STM32 là thiết bị giao tiếp vạn vật và ESP32 là IoT gateway. Tại VPS (network layer), hệ thống back-end là tập hợp của Message Queueing Telemetry Transport (MQTT) Broker, MongoDB database, và EggJS server, và back-end được triển khai bằng kỹ thuật Docker Containerizing.

Tại application layer, Nginx web-server được triển khai để cung cấp giao diện web trực quan hóa dữ liệu thiết bị cho người dùng. Cụ thể, giao diện người dùng là giao diện web được xây dựng trên cơ sở của nền tảng Ant Design Pro và có các thành phần giao diện có thể tùy chỉnh và kéo thả. Về giao tiếp và đồng bộ trong processing layer, hệ thống sử dụng cơ chế giao tiếp có các thành phần nhận diện frame và phát hiện lỗi. Từ đó, cơ chế giao tiếp có độ tin cậy và tự phục hồi, và triển khai giữa EggJS server, bo mạch, và giao diện web. Tiếp theo, cơ chế đồng bộ được triển khai dựa trên sự kiện "change stream" của MongoDB và giữa EggJS server, bo mạch, và giao diện web. Cuối cùng, cũng trong processing layer, mật mã hóa nhẹ ChaCha20-Poly1305 được sử dụng trên giao tiếp giữa IoT gateway ESP32 và EggJS server.

1.4 Đóng góp

Kết quả của khóa luận này là những đóng góp trong kỹ thuật giao tiếp-đồng bộ, phát triển và triển khai hệ thống back-end và giao diện web. Trong giao tiếp-đồng bộ, kỹ thuật giao tiếp-đồng bộ đã được triển khai trên thiết bị STM32, gateway ESP32, và API server. Trong mật mã hóa, kỹ thuật mật mã hóa nhẹ ChaCha20-Poly1305 đã áp dụng trên giao tiếp giữa gateway ESP32 và API server. Trên hệ thống back-end, MQTT broker và API server được phát triển trên nền tảng NodeJS. Hơn nữa, kỹ thuật Docker containerizing được sử dụng để triển khai các thành phần của hệ thống back-end. Cuối cùng, về giao diện, giao diện được phát triển trên nền tảng Ant Design Pro và ngôn ngữ TypeScript, và được triển khai bằng máy chủ web Nginx.

1.5 Bố cục

Báo cáo khóa luận được chia thành 5 chương. Đó là:

- Chương 1: GIỚI THIỆU.
- Chương 2: CÁC HỆ THỐNG VÀ CÔNG NGHỆ LIÊN QUAN.
- Chương 3: THỰC HIỆN NỀN TẢNG IOT SỬ DỤNG HỆ THỐNG BACK-END GIAO TIẾP PHẦN CỨNG QUA WIFI KẾT HỢP KỸ THUẬT MẬT MÃ HÓA NHẸ.
- Chương 4: KẾT QUẢ.
- Chương 5: KẾT LUẬN.

Đầu tiên, chương 1 đưa ra các nội dung tổng quát của khóa luận. Các nội dung bao gồm việc đặt vấn đề, mục tiêu đề ra, giải pháp đề xuất, đóng góp, và bố cục của khóa luận.

Tiếp theo, chương 2 đưa ra các đánh giá và khảo sát các hệ thống IoT trong việc giải quyết vấn đề của khóa luận. Hơn nữa, chương này còn đưa ra mô hình hệ thống IoT được đề xuất và các lý thuyết xây dựng hệ thống IoT được đưa ra.

Trong việc triển khai hệ thống, chương 3 trình bày các phương pháp và giải pháp đề xuất đề xây dựng hệ thống IoT. Các nội dung này trình bày các phương pháp và giải pháp cho việc xây dựng mô hình, bo mạch, giao diện web, và hệ thống back-end.

Về các kết quả, chương 4 đưa ra các kết quả đã đạt được trên phần cứng, VPS, và giao diện web.

Cuối cùng, chương 5 đưa ra các mô tả ngắn gọn về mục tiêu, phương pháp, kết quả đạt được, ý nghĩa của kết quả cũng như những hạn chế, giới hạn của đề tài và hướng phát triển.

CHƯƠNG 2: CÁC HỆ THỐNG VÀ CÔNG NGHỆ LIÊN QUAN

Chương 2 đưa ra các khảo sát và đánh giá một số giải pháp IoT. Cụ thể, các giải pháp sử dụng nền tảng Blynk, ThingsBoard, và ESP32 web server chế độ Access Point sẽ được đánh giá và nhận định. Cuối cùng, mô hình hệ thống IoT được sử dụng để đưa ra giải pháp cho việc giám sát và điều khiển các trạm BTS online, sẽ được trình bày ở mục 2.4.

2.1 Hệ thống IoT sử dụng nền tảng Blynk

2.1.1 Tổng quan về Blynk

Blynk là một trong những nền tảng IoT được sử dụng nhiều nhất hiện nay. Nền tảng này cung cấp bộ ứng dụng toàn diện cho các hệ thống IoT của cá nhân, người dùng, và doanh nghiệp. Trên nền nảng này, bộ ứng dụng toàn diện của Blynk cho phép tạo mẫu (prototyping), phát triển, và quản lý từ xa các thiết bị điện tử được kết nối của tất cả phạm vi (scale) [1].

2.1.2 Những vấn đề có thể giải quyết thông qua nền tảng Blynk

Trong việc giám sát BTS, thiết kế, và triển khai hệ thống IoT, Blynk cung cấp giải pháp cho phần cứng, cấu hình, và giao diện.

Tại phần cứng, Blynk Library thư viện portable C++ dễ sử dụng và có thể chạy trên nhiều bo mạch. Cụ thể, thư viện này có thể cài đặt trên ESP32 IoT gateway của bo mạch của PTN DESLab. Hơn nữa, Blynk Library dễ dàng cấu hình khi sử dụng Blynk Edgent. Đây là giải pháp được thiết kế để đơn giản hóa các kết nối của các thiết bị tới Blynk cloud. Vì vậy, Blynk Library và Edgent là những giải pháp phần cứng để triển khai giao thức kết nối tới cloud trên bo mạch.

Về mặt cấu hình, Blynk Console là ứng dụng web nhiều tính năng phục vụ cho nhiều đối tượng người dùng. Ứng dụng này cung cấp các template có sẵn để giúp người dùng dễ dàng cấu hình General Setting, Datastream, Event, Notification, ... Dựa trên ứng dụng web Blynk Console, người dùng dễ dàng cấu hình các kênh dữ liệu cũng như kiểu dữ liệu của các Datastream. Từ đó, các cấu hình này tạo ra cơ sở dữ liệu lưu trữ các giá trị từ phần cứng và trực quan hóa các dữ liệu.

Về mặt giao diện, nên tảng Blynk hỗ trợ người dùng giám sát và điều khiển các thiết bị trên dashboard trên nền tảng web và mobile app. Dựa trên các dashboard, người dùng có thể dễ dàng quản lý, quan sát dữ liệu từ thiết bị. Hơn nữa, đây là

giao diện của no-code mobile app và web app, nên người dùng dễ dàng chỉnh sửa các widget trên giao diện hiển thị. Với các đặc tính của Blynk no-code editor, dashboard của hệ thống rất dễ dàng được thiết kế cho người dùng.

2.1.3 Vấn đề không thể giải quyết thông qua Blynk

Khi sử dụng Blynk, nền tảng này không cung cấp mã nguồn mở. Vì vậy, mật mã hóa không thể cài đặt trên giao tiếp giữa gateway và cloud. Đây là vấn đề lớn nhất dẫn đến việc hệ thống IoT không thể triển khai thông qua Blynk.

2.2 Hệ thống IoT sử dụng ThingsBoard

2.2.1 Tổng quan về ThingsBoard

ThingsBoard là một nền tảng IoT mã nguồn mở cho phép nhanh chóng phát triển, quản lý, và mở rộng của các dự án IoT [2]. Mục đích của ThingsBoard là cung cấp IoT cloud hoặc giải pháp tại chỗ (on-premises) một cách đột phá mà có thể cung cấp hạ tầng phía server cho các ứng dụng IoT.

2.2.2 Những vấn đề có thể giải quyết thông qua nền tảng ThingsBoard

Trong việc giám sát BTS, thiết kế, và triển khai hệ thống IoT, nền tảng ThingsBoard đưa ra giải pháp cho phần cứng, cấu hình, và giao diện.

Tại phần cứng, ThingsBoard Devices Library là tập hợp những hướng dẫn và đoạn code mà giải thích cách thức kết nối các IoT development board phổ biến tới nền tảng ThingsBoard [3]. Trên hệ thống IoT được đề xuất, gateway ESP32 có thể sử dụng thư viện C++ "Arduino ThingsBoard SDK". Thư viện này giúp gateway ESP32 có thể tương tác với ThingsBoard cloud thông qua các giao thức như MQTT, HyperText Transfer Protocol (HTTP)...

Về cấu hình, ThingsBoard web cho phép người dùng cấu hình các thiết bị trên trang "Entities → Devices". Từ đó, các thiết bị phần cứng có thể trao đổi dữ liệu với ThingsBoard cloud thông qua "Access Token".

Về giao diện, ThingsBoard web cho phép người dùng tạo các dashboard linh hoạt trên trang "Dashboards". Trên trang này, người dùng có thể chọn các widget để hiển thị các giá trị của bo mạch phần cứng.

2.2.3 Vấn đề ThingsBoard không thể giải quyết

Khác với Blynk, ThingsBoard là nền tảng IoT mã nguồn mở cung cấp giải pháp hạ tầng phía server. Vì vậy, hệ thống IoT được đề xuất, có thể host và tùy chỉnh ThingsBoard server trên VPS. Điều này có thể giúp hệ thống IoT cài đặt kỹ thuật mật mã hóa tùy chỉnh trên ThingsBoard server. Tuy nhiên, kích thước mã nguồn của ThingsBoad server là quá lớn và VPS của các quản trị viên không thể host ứng dụng server này. Vì vậy, kích thước mã nguồn quá lớn so với khả năng của VPS là nguyên nhân không thể sử dụng ThingsBoard server trên hệ thống IoT.

2.3 Các hệ thống IoT đơn giản sử dụng ESP32 web server chế độ Access Point

2.3.1 Những vấn đề có thể giải quyết khi sử dụng ESP32 web server chế độ Access Point

Trong việc giám sát BTS, thiết kế, và triển khai hệ thống IoT, việc sử dụng ESP32 web server chế độ Access Point, có thể giải quyết vấn đề phần cứng, thiết kế, và triển khai hệ thống IoT tại một BTS.

Về phần cứng, MCU ESP32 có thể được lập trình như một thiết bị và web server. Cụ thể, MCU ESP32 có thể giao tiếp với các ngoại vi và cảm biến thông qua các General-purpose input/output (GPIO).

Về cấu hình, hệ thống dựa trên mô hình này, chỉ sử dụng MCU ESP32. Do đó, MCU ESP32 có thể được cấu hình cơ sở dữ liệu một cách thủ công và triển khai kỹ thuật mật mã hóa nhẹ bằng việc lập trình firmware đơn thuần.

Về giao diện, MCU ESP32 được cấu hình như một web server. Do đó, MCU này cung cấp một giao diện tĩnh hiển thị các tương tác phục vụ cho việc hiển thị và điều khiển các giá trị trên cơ sở dữ liệu của nó.

2.3.2 Những vấn đề không thể giải quyết khi sử dụng ESP32 web server chế độ Access Point

Hệ thống IoT sử dụng ESP32 web server chế độ Access Point không thể giải quyết vấn đề kích thước hệ thống và giám sát hệ thống từ xa.

Đầu tiên, về vấn đề kích thước hệ thống, các quản trị viên VNPT muốn giám sát nhiều BTS một cách tập trung. Khi đó, một MCU ESP32 không đủ dung lượng để lưu trữ dữ liệu từ nhiều BTS cũng như dữ liệu người dùng.

Cuối cùng, khi sử dụng mô hình ESP32 web server, người dùng chỉ có thể truy cập giao diện web khi có cùng Access Point với MCU ESP32. Vì vậy, khi người dùng giám sát từ xa và không có cùng Access Point với ESP32, người dùng không thể giám sát thông số của các BTS.

2.4 Mô hình hệ thống IoT sử dụng bo mạch của PTN DESLab giao tiếp với VPS

Đây là mô hình hệ thống IoT được áp dụng để giám sát các trạm BTS. Tại PTN DESLab, mô hình này đã có kết quả để chạy và thử nghiệm trong thời gian ngắn hạn tại BTS của các quản trị viên VNPT. Tuy nhiên, mô hình này vẫn phải đưa ra các khái niệm và kỹ thuật cốt lõi để có thể phát triển lâu dài. Trong khóa luận tốt nghiệp này, các khái niệm và kỹ thuật cốt lõi cần có, sẽ được trình bày và xây dựng để hệ thống IoT hiện tại, có thể tiếp tục phát triển. Các khái niệm cỗi lõi bao gồm mô hình-sơ đồ của hệ thống IoT, giao thức frame, cơ chế đồng bộ, cách thức triển khai mật mã hóa, và cách thức triển khai hệ thống back-end và giao diện.

2.4.1 Mô hình và sơ đồ của hệ thống IoT

Mô hình của hệ thống IoT cần được xây dựng, là mô hình IoT 4 lớp (minh họa như hình 2.1. Cụ thể, mô hình bao gồm sensing layer, network layer, data processing layer, và application layer.

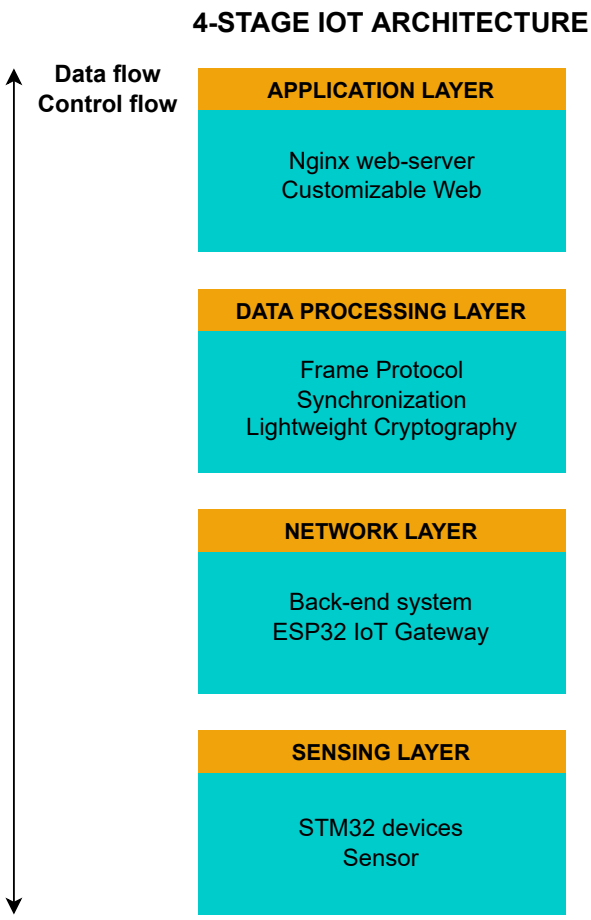
2.4.1.1 Sensing layer

Đây là lớp đầu tiên của hệ thống IoT và sensing layer chứa các thiết bị vật lý, cảm biến, và ngoại vi trên bo mạch. Hoạt động của tầng này chủ yếu là thu thập và xử lý dữ liệu từ các cảm biến và ngoại vi, sau đó gửi dữ liệu lên tầng network.

Trong sensing layer, thiết bị sử dụng để giao tiếp với cảm biến và ngoại vi, là MCU STM32. Các giao thức sử dụng trong tầng này là các giao thức có dây như UART, SPI, I2C,...

2.4.1.2 Network layer

Network layer là lớp chứa các thiết bị thu nhận và truyền tải dữ liệu trên Internet. Dựa trên lớp này, dữ liệu sẽ được luân chuyển từ IoT gateway đến các ứng dụng back-end.



Hình 2.1: Mô hình 4 lớp của hệ thống IoT. Nguồn tham khảo InterviewBit [4]

Trong network layer, IoT gateway là MCU ESP32 trên bo mạch. Gateway sẽ định tuyến dữ liệu từ hệ thống back-end đến sensing layer và ngược lại. Cuối cùng, hệ thống back-end thu nhận, xử lý, và lưu trữ dữ liệu từ sensing layer và đồng bộ các dữ liệu này với cơ sở dữ liệu của thiết bị vật lý và giao diện người dùng.

2.4.1.3 Data processing layer

Data processing layer là lớp chứa các kỹ thuật xử lý dữ liệu. Trong hệ thống này, data processing layer sẽ phân giải frame, mã hóa/giải mã dữ liệu, và đồng bộ dữ liệu với sensing và application layer.

2.4.1.4 Application layer

Application layer là lớp cung cấp ứng dụng của hệ thống thông qua các tương tác người dùng. Tại lớp này, hệ thống IoT sẽ cung cấp giao diện web tùy chỉnh cho người dùng. Từ đó, người dùng có thể giám sát và điều khiển dữ liệu của sensing và network layer.

2.4.2 Giao thức frame

2.4.2.1 Tổng quan về giao thức frame

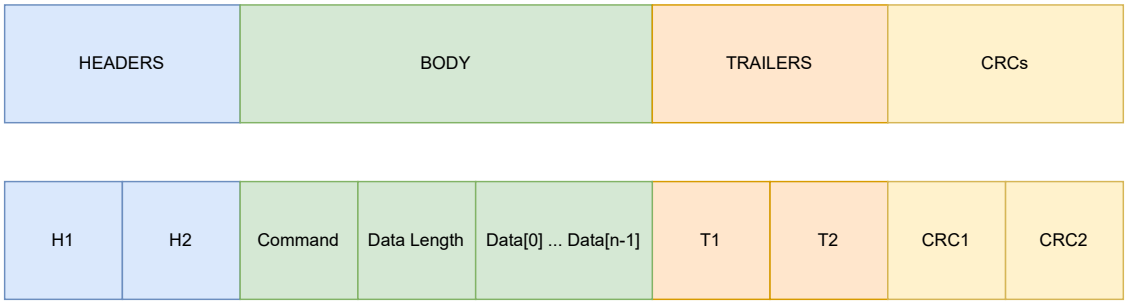
Giao thức frame được định nghĩa để tạo ra cấu trúc frame trong giao tiếp device-gateway. Cấu trúc frame này là duy nhất và khác biết với các hệ thống khác. Điều này giúp việc giao tiếp trên bo mạch dễ triển khai và có tính bảo mật.

Trên bo mạch, giao thức frame giúp giao tiếp device-gateway có tính bảo mật và đáng tin cậy. Đầu tiên, hệ thống sẽ có cấu trúc frame duy nhất, riêng tư, và bảo mật. Vì vậy, cấu trúc frame này không thể phân giải trên các hệ thống khác. Tiếp theo, vì giao tiếp device-gateway có thể bị lỗi do nhiễu trên bo mạch, nên cấu trúc frame cung cấp các byte CRC để phát hiện lỗi khi giao tiếp diễn ra. Điều này đảm bảo hệ thống vẫn hoạt động khi phát sinh lỗi trong giao tiếp device-gateway.

2.4.2.2 Cấu trúc frame

Cấu trúc frame (minh họa như hình 2.2) được thiết kế bằng cách bố trí thêm các byte header, trailer, và CRC vào dữ liệu truyền. Cụ thể, headers chứa 2-byte h1 và h2; trailers chứa 2-byte t1 và t2; CRC chứa 2-byte CRC1 và CRC2 để tính toán CRC 16-bit; và phần body chứa 1-byte command, 2-byte data length, và các byte data.

Để nhận diện frame, các header và trailer dùng để nhận diện phần đầu và cuối frame. Tiếp theo, 1-byte command, 2-byte data length, và các byte data để hệ thống thực thi lệnh với dữ liệu tương ứng của frame. Cuối cùng, 2-byte CRC dùng để soát lỗi trên phần body của frame.



Hình 2.2: Cấu trúc frame của hệ thống.

Để phân giải frame, kỹ thuật phân giải frame (minh họa như hình 2.3) được triển khai trên giao tiếp UART của MCU STM32 và gateway ESP32. Với mỗi byte nhận được, chỉ số của frame sẽ được kiểm tra. Với mỗi chỉ số frame, frame sẽ được kiểm tra header, trailer, CRC, hoặc thu thập các dữ liệu phần body. Cuối cùng, quá trình phần giải thành công khi việc kiểm tra các header, trailer và tính toán CRC thành công và quá trình thất bại khi việc kiểm tra header, trailer, và CRC thất bại; hoặc timeout.

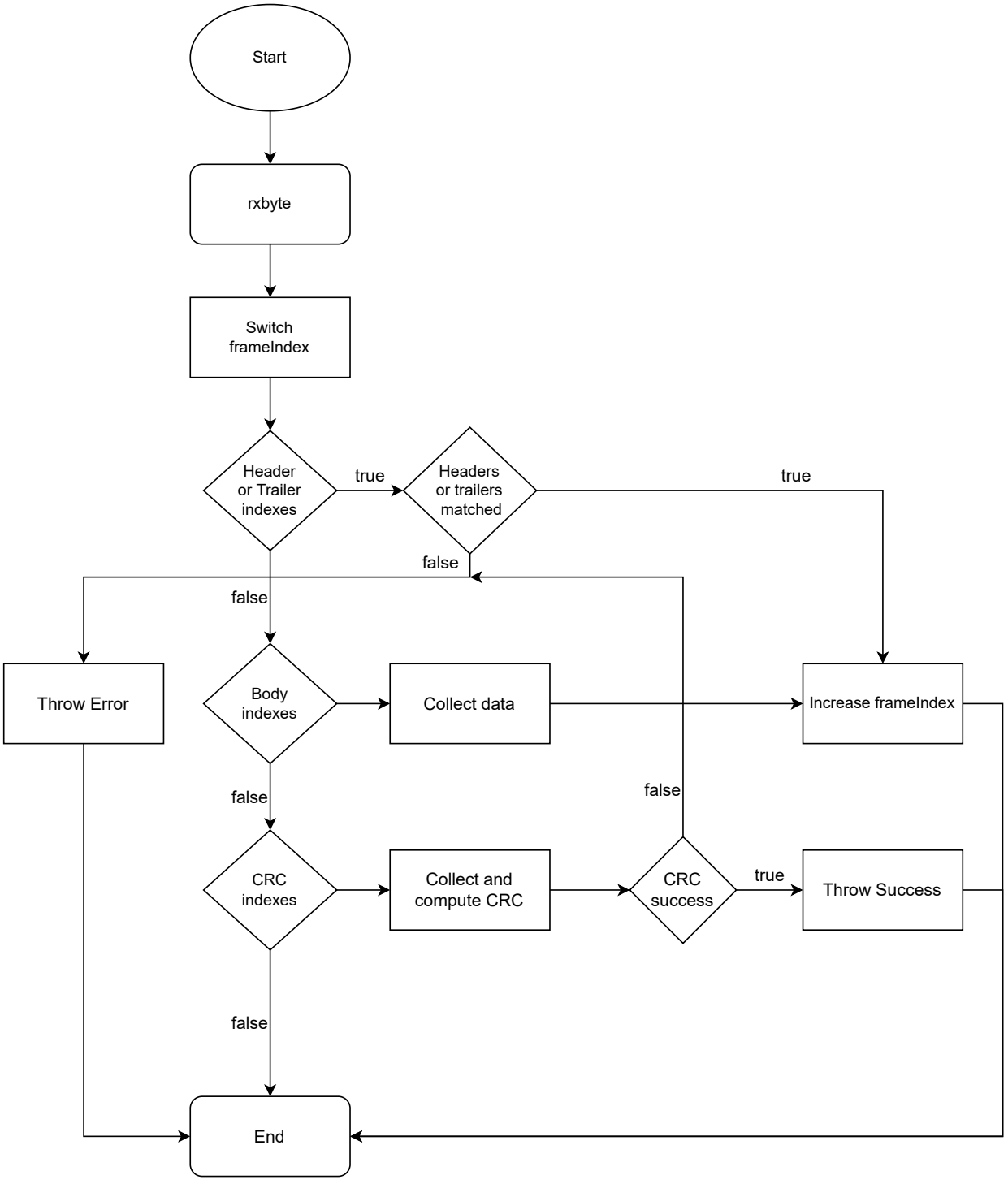
2.4.3 Cơ chế đồng bộ

Cơ chế đồng bộ (minh họa như hình 2.4) được thiết kế dựa trên các yêu cầu cập nhật database. Với mỗi yêu cầu cập nhật dữ liệu của database, database phải thông báo sự kiện cập nhật tới API server. Thông qua sự kiện cập nhật từ database, API server sẽ đồng bộ dữ liệu được cập nhật với bo mạch và giao diện. Giao diện sẽ được đồng bộ trực tiếp qua Internet và thiết bị được đồng bộ gián tiếp qua IoT gateway.

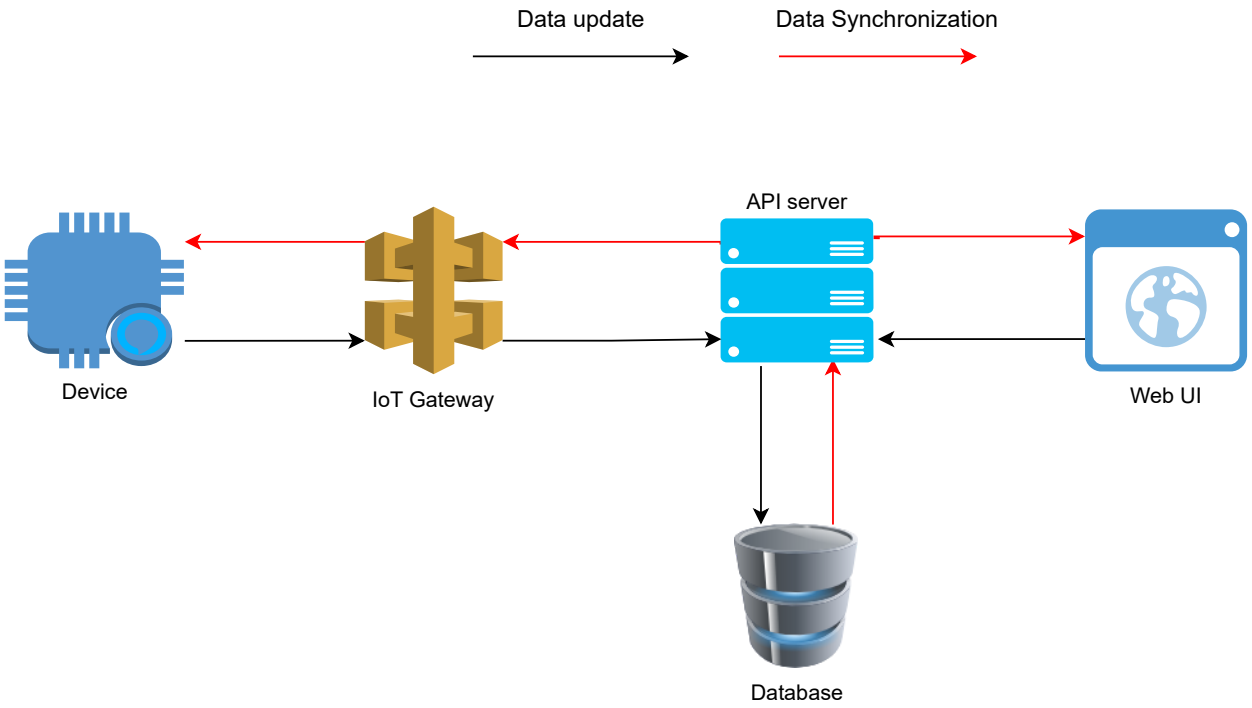
Trên hệ thống hiện tại, model của các Virtual Storage (VS) sẽ được đồng bộ khi có sự thay đổi dữ liệu của model này trên database. Khi có sự thay đổi dữ liệu model của VS, dữ liệu model của VS sẽ được đồng bộ tới thiết bị và giao diện thông qua cách thức hoạt động của cơ chế đồng bộ được đề cập.

2.4.4 Giao diện web linh hoạt

Trong các hệ thống IoT, cấu hình của các thiết bị luôn có thể thay đổi. Vì vậy, giao diện hiển thị cũng phải có tính linh hoạt và có thể tùy chỉnh để đáp ứng các



Hình 2.3: Flowchart minh họa kỹ thuật phân giải frame.



Hình 2.4: Cơ chế đồng bộ của hệ thống IoT.

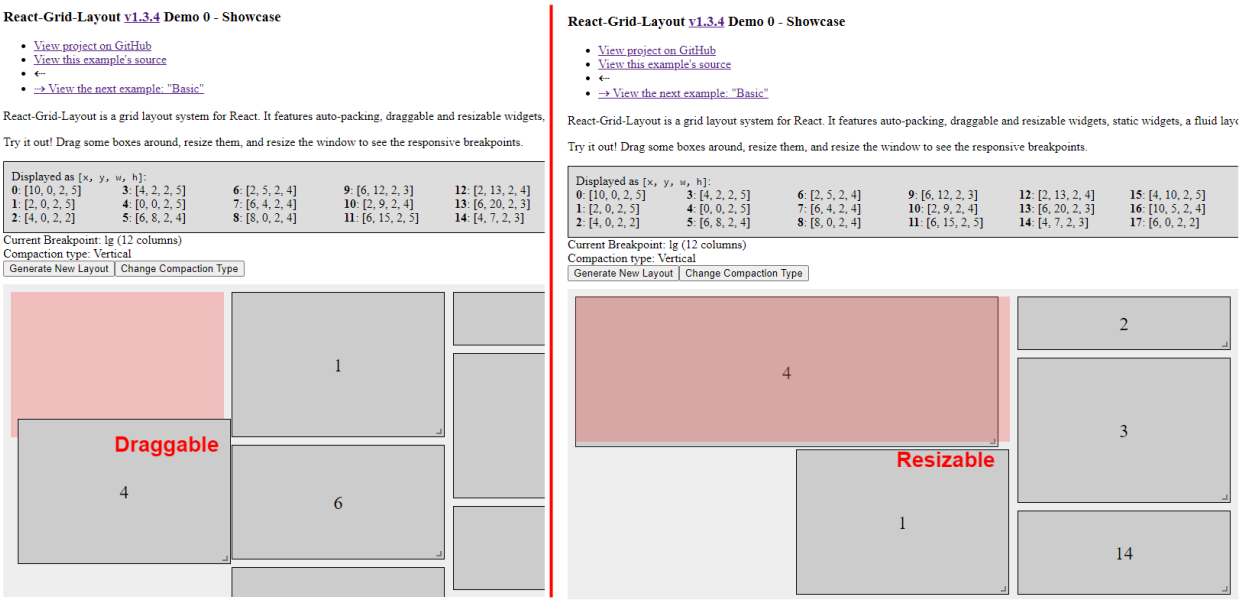
thay đổi đó. Như vậy, giao diện có thể thay đổi tương ứng với cấu hình của các thiết bị. Hơn nữa, đối với người dùng, người dùng có thể dễ dàng thay đổi giao diện trong khi họ không cần phải lập trình.

Giao diện web linh hoạt của hệ thống được phát triển trên framework ReactJS. Dựa trên framework này, thư viện "React-Grid-Layout" sẽ được sử dụng để phát triển giao diện linh hoạt. Khi sử dụng thư viện này, một hệ thống layout dạng lưới (minh họa như hình 2.5) sẽ được tạo ra. Trên mạng lưới đó, người dùng có thể kéo thả và thay đổi kích thước của các widget.

2.4.5 Mật mã hóa nhẹ ChaCha20-Poly1305

Mục này sẽ trích xuất và dịch thuật nội dung của RFC 7359 [5].

ChaCha20-Poly1305 là mật mã hóa xác thực với dữ liệu liên kết Authenticated Encryption with Associated Data (AEAD). Đây là sự kết hợp của hai thuật toán độc lập. Thuật toán đầu tiên là ChaCha20, đây là thuật toán mã hóa dòng tốc độ cao. ChaCha20 được cho là nhanh hơn đáng kể so với thuật toán Advanced Encryption Standard (AES) trên môi trường software-only. Hơn nữa, ChaCha20 chạy nhanh hơn AES ba lần trên những nền tảng không hỗ trợ phần cứng AES. Về thuật toán Poly1305, Poly1305 là thuật toán xác thực tin nhắn tốc độ cao (high-speed message



Hình 2.5: Hệ thống layout dạng lưới của React-Grid-Layout.

authentication code). Việc triển khai thuật toán này cũng rất dễ dàng và có độ chính xác cao.

2.4.5.1 ChaCha Quarter Round

Phép tính cơ bản của thuật toán ChaCha là quarter round. Nó tính toán dựa trên 4 số nguyên không dấu 32-bit, ký hiệu là a, b, c, và d. Việc tính toán diễn ra tuần tự như sau:

1. a += b; d ∧= a; d <<<= 16;
2. c += d; b ∧= c; b <<<= 12;
3. a += b; d ∧= a; d <<<= 8;
4. c += d; b ∧= c; b <<<= 7;

Trong đó, dấu "+" ký hiệu cho phép cộng modulo 32-bit, "∧" ký hiệu cho toán tử bit Exclusive OR (XOR), và "<<< n" ký hiệu cho n-bit dịch vòng bên trái (left rotation).

Về ChaCha state, ChaCha state là tập hợp của 16 số nguyên không dấu 32-bit, được biểu diễn dưới dạng ma trận 4x4 (như bảng 2.1). Vì vậy, với mỗi phép toán quarter round, phép toán chỉ được thực hiện trên 4 trong số 16 số nguyên của ChaCha state.

Bảng 2.1: Mô tả bố cục ChaCha state ở dạng ma trận.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

2.4.5.2 ChaCha20 Block Function

ChaCha20 Block Function là quá trình thực hiện việc biến đổi một ChaCha state bằng việc thực hiện nhiều lần các quarter round.

Đầu vào của khối ChaCha20 bao gồm:

- Một key 256-bit, là sự kết hợp của 8 số nguyên 32-bit little-endian.
- Một nonce 96-bit, là sự kết hợp của 3 số nguyên 32-bit little-endian.
- Một khối đếm (block count parameter), là một số nguyên 32-bit little-endian.

Đầu ra của khối là 64 byte ngẫu nhiên.

Về việc khởi tạo ChaCha State, ChaCha State được khởi tạo bằng cách bố trí các tham số đầu vào của ChaCha20 dưới dạng ma trận 4x4 (minh họa như bảng 2.2) như sau:

- 4 word đầu tiên (từ vị trí 0→3) là các constant: 0x61707865, 0x3320646e, 0x79622d32, 0x6b206574.
- 8 word tiếp theo (từ vị trí 4→11) được trích xuất từ 256-bit key bằng việc đọc các byte theo trật tự little-endian, đọc theo từng khối, mỗi khối là 1 word.
- Word thứ 12 là block counter. Vì mỗi ChaCha Block có độ dài 64 byte, một word 32-bit đủ cho 256 gigabytes dữ liệu.
- Các word thứ 13→15 là nonce, cùng một key thì không nên lặp lại nonce. Word thứ 13 là 32 bit đầu tiên của đầu vào nonce như một số nguyên little-endian, trong khi đó, word thứ 15 là 32 bit còn lại.

Trong đó, c=constant, k=key, b=blockcount, và n=nonce

Trong ChaCha20 Block Function, ChaCha20 chạy 20 round, xen kẽ giữa "column rounds" và "diagonal rounds". Mỗi round bao gồm bốn quarter-round, được minh

Bảng 2.2: Ma trận khởi tạo của ChaCha state.

cccccccc	cccccccc	cccccccc	cccccccc
kkkkkkkk	kkkkkkkk	kkkkkkkk	kkkkkkkk
kkkkkkkk	kkkkkkkk	kkkkkkkk	kkkkkkkk
bbbbbbbb	nnnnnnnn	nnnnnnnn	nnnnnnnn

họa như trình tự bên dưới. Trong một round của ChaCha20 Block Function, các quarter-round 1→4 là phần thuộc "column" round, và 5→8 là phần thuộc "diagonal" round:

1. QUARTERROUND(0, 4, 8, 12)
2. QUARTERROUND(1, 5, 9, 13)
3. QUARTERROUND(2, 6, 10, 14)
4. QUARTERROUND(3, 7, 11, 15)
5. QUARTERROUND(0, 5, 10, 15)
6. QUARTERROUND(1, 6, 11, 12)
7. QUARTERROUND(2, 7, 8, 13)
8. QUARTERROUND(3, 4, 9, 14)

Sau khi kết thúc 20 round (hoặc 10 lần thực hiện 8 trình tự ở trên), các word đầu vào ban đầu được cộng với các word đầu ra. Sau đó, kết qua được serialize bằng việc sắp xếp các từng word một theo trình tự little-endian.

2.4.5.3 Thuật toán mã hóa ChaCha20

ChaCha20 là mã hóa dòng được thiết kế bởi D. J. Bernstein. Nó là sự cải tiến của thuật toán Salsa20, và sử dụng 256-bit key.

ChaCha20 liên tục thực thi ChaCha20 Block Function, với cùng key và nonce, và với việc liên tục tăng block counter. Sau đó, ChaCha20 sẽ tuần tự hóa (serialize) kết quả của ChaCha state bằng việc ghi các số theo thứ tự little-endian, tạo keystream block.

Đầu vào của thuật toán ChaCha20 bao gồm

- Một key 256-bit.
- Một counter 32-bit ban đầu. Counter có thể thiết lập bất cứ giá trị nào, nhưng thường là 0 hoặc 1. Điều này có nghĩa khi dùng một lần nếu ta sử dụng khối couter 0 cho việc gì đó, ví dụ như việc tạo ra one-time authenticator key như là một phần của thuật toán AEAD.
- Một nonce 96-bit. Trong một số giao thức, nonce được xem như là một vector khởi tạo.
- Một plaintext có độ dài tùy ý.

Đầu ra của thuật toán là tin nhắn được mã hóa, hoặc "ciphertext", có cùng độ dài.

Việc giải mã được tiện hành tương tự như mã hóa. ChaCha20 Block Function được sử dụng để tạo keystream, nó được dùng để XOR với ciphertext để khôi phục lại plaintext.

2.4.5.4 Thuật toán Poly1305

Poly1305 là thuật toán tạo xác thực một lần (one-time authenticator) được thiết kế bởi D. J. Bernstein. Poly1305 dùng 32-byte one-time key và một message, và cho ra một xác thực 16-byte tag. Tag này dùng để xác thực message.

Bất kể key được tạo ra như thế nào, key được phân vùng thành hai phần, được gọi là "r" và "s". Cặp (r,s) nên là duy nhất, và bắt buộc không thể dự đoán trước trong mỗi lần tạo, "r" có thể tùy chỉnh là một hằng số.

"s" nên được tạo ra có tính chất không thể dự đoán trước, nhưng hoàn toàn chấp nhận việc tạo ra cả "r" và "s" một cách duy nhất tại một thời điểm. Bởi vì cả "r" và "s" đều có độ dài 128 bit, nên tạo ra chúng một cách ngẫu nhiên thì cũng chấp nhận được.

Đầu vào của thuật toán Poly1305 bao gồm:

- Một one-time key 256-bit.
- Một message có độ dài tùy ý.

Đầu ra của thuật toán là 128-bit tag.

Thuật toán Poly1305 diễn ra tuần tự như sau. Đầu tiên, giá trị của "r" được kẹp chặt (clamp).

Tiếp đến, thiết lập hằng số nguyên số "P" là $2^{135} - 5$: 3fffffffffffffffffffffffffffffffb. Sau đó ta cũng thiết lập biến "accumulator" bằng 0.

Sau đó, chia message thành các khối 16-byte. Khối cuối cùng có thể có độ dài ngắn hơn:

- Đọc khối message như từng số little-endian.
- Thêm một bit ở vị trí ngoài cùng của các octet. Với một khối 16-byte, điều này tương đương với việc cộng với 2^{128}. Khối có độ dài nhỏ hơn 10-byte, nó có thể là 2^{120}, 2^{112}, hoặc bất cứ số là lũy thừa của 2 mà nó có thể chia hết cho 8, nhỏ nhất là 2^8.
- Nếu block không có độ dài 17 byte (ở block cuối cùng), thêm các bit 0 (pad) vào block. Điều này là vô nghĩa nếu ta coi các block như các số nguyên.
- Cộng số này với "accumulator".
- Nhân "accumulator" với "r".
- Thiết lập "accumulator" bằng phép chia modulo cho "P".
  Tóm lại: $accumumator = ((accumulator + block) * r)\%P$.

Cuối cùng, giá trị của secret key ("s") được cộng vào "accumulator", và 128 least significant bit được tuần tự hóa theo trật tự little-endian.

2.4.5.5 Tạo Poly1305 key sử dụng ChaCha20

Như đã đề cập ở mục 2.4.5.4, có thể chấp nhận việc tạo one-time key cho Poly1305 một cách giả ngẫu nhiên (pseudorandomly). Ở mục này, phương pháp được đề xuất là tạo one-time key cho Poly1305 sử dụng ChaCha20.

Để tạo cặp key (r,s), chúng ta sử dụng ChaCha20 Block Function. Phương pháp được đề xuất là gọi ChaCha20 Block Function với các tham số đầu vào như sau:

- 256-bit key từ keystream sinh ra từ ChaCha20 Block Function.
- Block counter thiết lập bằng 0.
- Một nonce 96-bit.

Sau khi chạy ChaCha20 Block Function, nó sinh ra 512-bit state. Lấy 256 bit đầu tiên hoặc state tuần tự hóa (serialized state), và sử dụng chúng như one-time Poly1305 key: 128 bit đầu tiên giữ lại và hình thành nên "r", và 128 bit còn lại trở thành "s". 256 bit còn lại bị loại bỏ.

2.4.5.6 Cấu trúc AEAD

Cấu trúc AEAD cho thuật toán ChaCha20-Poly1305 là mã hóa xác thực với dữ liệu liên kết. Các input cho AEAD_CHACHA20_POLY1305 là:

- Một key 256-bit.
- Một nonce 96-bit.
- Một plaintext có độ dài tùy ý.
- Dữ liệu xác thực liên kết với độ dài tùy ý - Arbitrary length additional authenticated data (AAD).

Các thuật toán gốc của ChaCha20 và Poly1305 được kết hợp thành AAD có đầu vào là một key 256-bit và nonce 96-bit như sau:

- Đầu tiên, Poly1305 one-time key được tạo ra từ key 256-bit và nonce sử dụng phương pháp ở mục 2.4.5.5.
- Tiếp theo, function, thực hiện mã hóa ChaCha20, được thực thi để mã hóa plaintext, sử dụng cùng key và nonce, và với couter được thiết lập bằng 1.
- Cuối cùng, Poly1305 function được thực thi với đầu vào là Poly1305 key được tính toán ở trên, và một message (plaintext).

Output từ AEAD là hai thành phần sau:

- Một ciphertext có cùng độ dài với plaintext.
- Một tag 128-bit, là output của Poly1305 function.

Việc giải mã là tương tự với những khác biệt sau:

- Các vai trò của ciphertext và plaintext được hoán đổi, nên function mã hóa ChaCha20 được áp dụng cho ciphertext, cung cấp plaintext.
- Poly1305 function vẫn thực thi dựa trên AAD và cipher text, không phải plaintext.
- Tag được tạo ra, được so sánh tag nhận được. Message được xác thực khi và chỉ khi các tag trùng khớp với nhau.

CHƯƠNG 3: THỰC HIỆN NỀN TẢNG IOT SỬ DỤNG HỆ THỐNG BACK-END GIAO TIẾP PHẦN CỨNG QUA WIFI KẾT HỢP KỸ THUẬT MẬT MÃ HÓA NHẸ

Chương 3 mô tả chi tiết phương pháp thực hiện và giải pháp đề xuất trên mô hình hệ thống IoT. Các mô tả này bao gồm mô hình, bo mạch, giao diện web, và hệ thống back-end.

3.1 Mô hình hệ thống IoT

Mô hình hệ thống IoT được đề xuất (minh họa như hình 3.1), được triển khai dựa trên sự kết hợp của bo mạch, hệ thống back-end, giao diện web, và giao thức kết nối giữa chúng. Dựa trên các thành phần này, mô hình IoT 4 lớp (được đề cập ở mục 2.4.1) sẽ được áp dụng. Trên mô hình này, bo mạch của PTN DESLab và VPS của các quản trị viên VNPT là hai thành phần chủ đạo.

Trên bo mạch, bo mạch được thiết kế dựa trên kết nối của các MCU STM32 và ESP32. Trong đó, MCU STM32 là thiết bị giao tiếp vạn vật (things) và thuộc sensing layer. Tiếp theo, MCU ESP32 là IoT gateway định tuyến các luồng dữ liệu trong giao tiếp sensing-network layer và thuộc lớp network layer.

Trên VPS, hệ thống IoT triển khai hệ thống back-end và Nginx web server. Cụ thể, hệ thống back-end bao gồm MQTT Broker, EggJS API server, và MongoDB database. Cuối cùng, Nginx web server cung cấp giao diện người dùng và giao diện người dùng được phát triển trên nền tảng ReactJS và giải pháp Ant Design Pro.

Trên hệ thống IoT, hệ thống có 3 kỹ thuật xử lý dữ liệu. Đó là giao thức frame, mật mã hóa nhẹ ChaCha20-Poly1305, và đồng bộ. Trong đó, giao thức frame được triển khai trên thiết bị STM32, ESP32 gateway, và API server. Tiếp theo, mật mã hóa ChaCha20-Poly1305 được sử dụng trên ESP32 gateway và API server. Cuối cùng, cơ chế đồng bộ được triển khai trên API server, giao diện web, và thiết bị STM32.

Về các giao thức truyền, hệ thống sử dụng giao thức có dây Universal Asynchronous Receiver-Transmitter (UART); các giao thức không dây: MQTT, Hypertext Transfer Protocol Security (HTTPS), và Websocket; và Virtual Local Area Network (VLAN). Trong đó, UART được sử dụng trong giao tiếp gateway-device; MQTT được sử dụng trong giao tiếp gateway-MQTT Broker và MQTT Broker-API server; HTTPS và Websocket được sử dụng trong giao tiếp API server-web; và VLAN được

sử dụng trong giao tiếp API server-database.

3.2 Bo mạch phần cứng

Bo mạch phần cứng được sử dụng để xây dựng hệ thống IoT (minh họa ở hình 3.2), là bo mạch phát triển IoT của PTN DESLab. Bo mạch này bố trí hai MCU STM32F407VGT6 và ESP Wroom 32 kèm theo các ngoại vi. Các ngoại vi bao gồm led, nút nhấn, switch, màn hình OLED, và các GPIO của hai MCU.

3.2.1 ESP32 IoT gateway

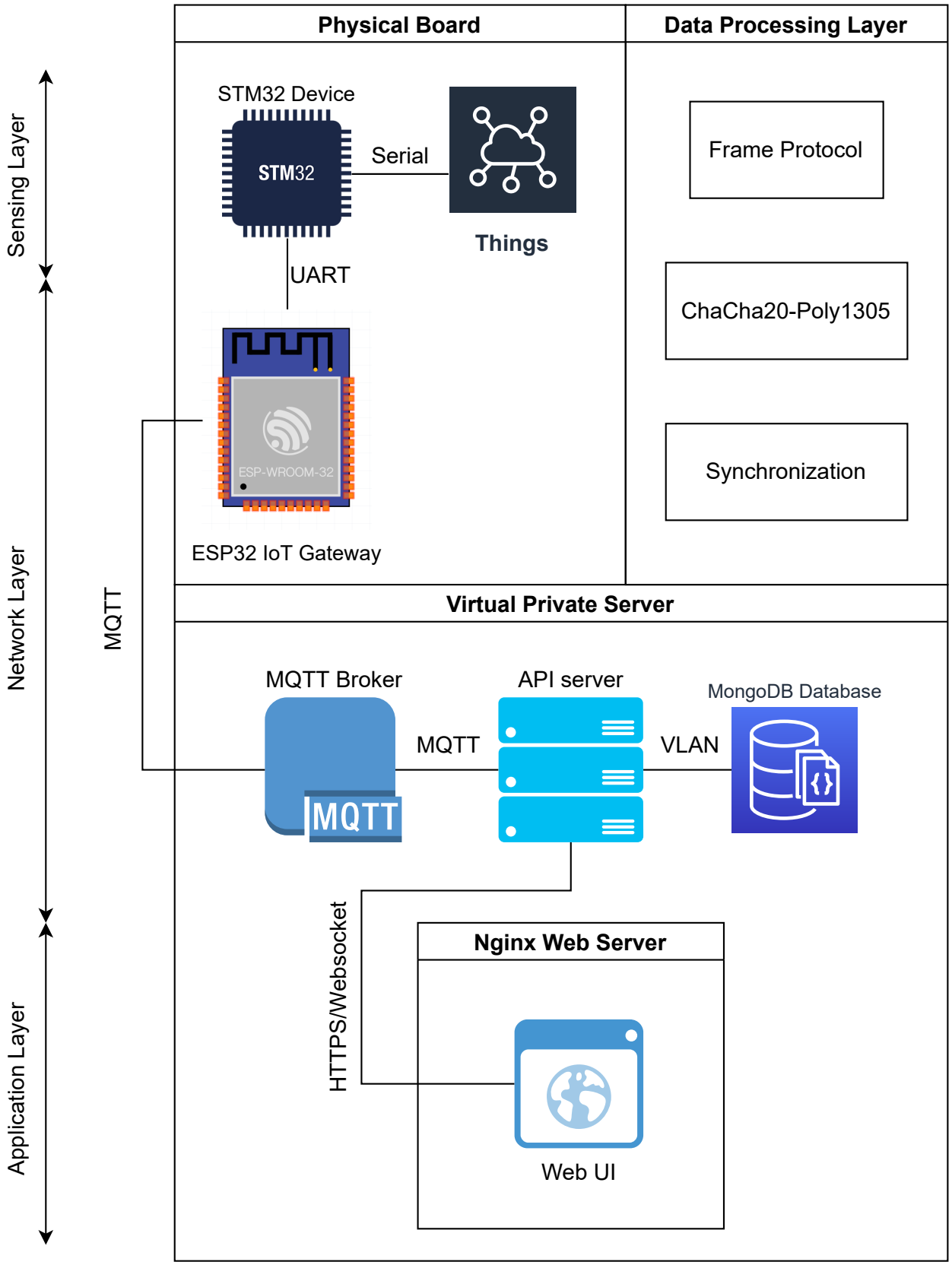
MCU ESP32 được phát triển trên framework Arduino, platform PlatformIO, và ngôn ngữ C++. ESP32 được bố trí trên bo mạch phát triển và có sẵn module WiFi (Wi-Fi: 802.11 b/g/n/e/i, 2.5 GHz). Vì vậy, MCU ESP32 được phát triển để được sử dụng như một IoT gateway. Về cơ bản, ESP32 IoT gateway có nhiệm vụ định tuyến các luồng dữ liệu trong giao tiếp STM32 device-API server và ngược lại. Để thực hiện được công việc này, MCU ESP32 phải thực hiện công việc phân giải và xử lý frame giao tiếp cũng như đệm dữ liệu (buffering data).

Về việc đệm dữ liệu, MCU ESP32 sử dụng bộ đệm vòng (circular buffer) có kích thước cố định để lưu trữ tạm thời các dữ liệu nhận được. Trên bo mạch hiện tại, ESP32 có hai bộ đệm vòng để nhận dữ liệu qua giao tiếp UART từ MCU STM32 và qua giao tiếp MQTT từ MQTT Broker. Cụ thể, bộ đệm vòng UART nhận dữ liệu từ ngắt UART và bộ đệm vòng MQTT nhận dữ liệu từ MQTT message callback. Dựa trên các buffer này, ESP32 tiến hành phân xử từng buffer (minh họa như hình 3.3) trong vòng lặp vô hạn của firmware. Để phân xử hợp lý cho từng bộ đệm, buffer selector được triển khai để lựa chọn buffer trong mỗi vòng lặp. Việc này giúp các buffer luôn được kiểm tra và mỗi buffer sẽ được tiến hành xý lý khi nó có dữ liệu.

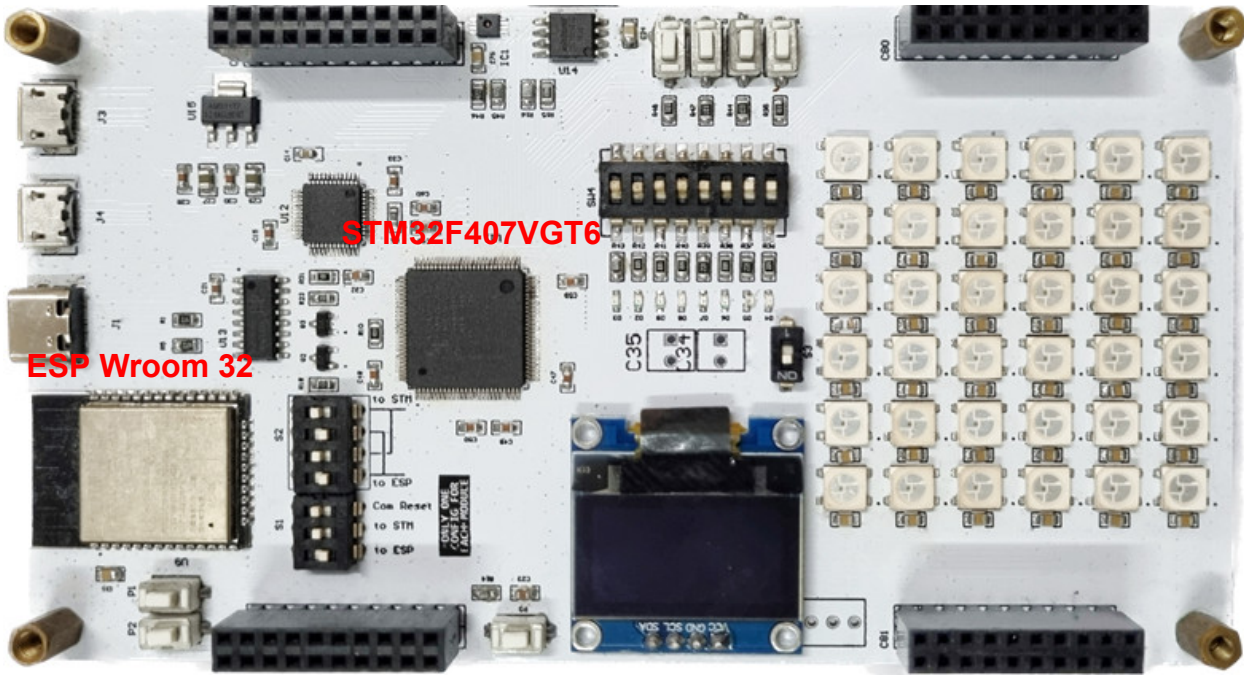
Về quá trình phân giải và xử lý frame, quá trình này diễn ra để xử lý dữ liệu trong bộ đệm vòng. Khi bộ đệm có dữ liệu, quá trình phân giải frame sẽ diễn ra trong một process độc lập (minh họa như hình 3.4). Process này sẽ kết thúc khi frame được phân giải thành công, thất bại, hoặc timeout.

3.2.2 STM32 device

MCU STM32 được phát triển trên công cụ STM32CubeIDE và sử dụng ngôn ngữ C. STM32 được phát triển như một thiết bị vật lý trong mô hình IoT. Cụ thể, MCU



Hình 3.1: Mô hình hệ thống IoT được đề xuất.



Hình 3.2: Bo mạch phát triển IoT của PTN DESLab.

STM32 có nhiệm vụ tương tác với các cảm biến và ngoại vi cũng như đồng bộ dữ liệu với API server.

STM32 thực hiện đệm dữ liệu, xử lý và phân giải frame như ESP32 gateway (được đề cập trong mục 3.2.1). Ngoài ra, STM32 có thể đọc và thay đổi cơ sở dữ liệu trên VPS thông qua các API. Các API này giúp STM32 gửi yêu cầu đọc ghi dữ liệu và đồng bộ với cơ sở dữ liệu. Các API này bao gồm API đọc dữ liệu của VS; và ghi dữ liệu VS theo kiểu Integer và Float.

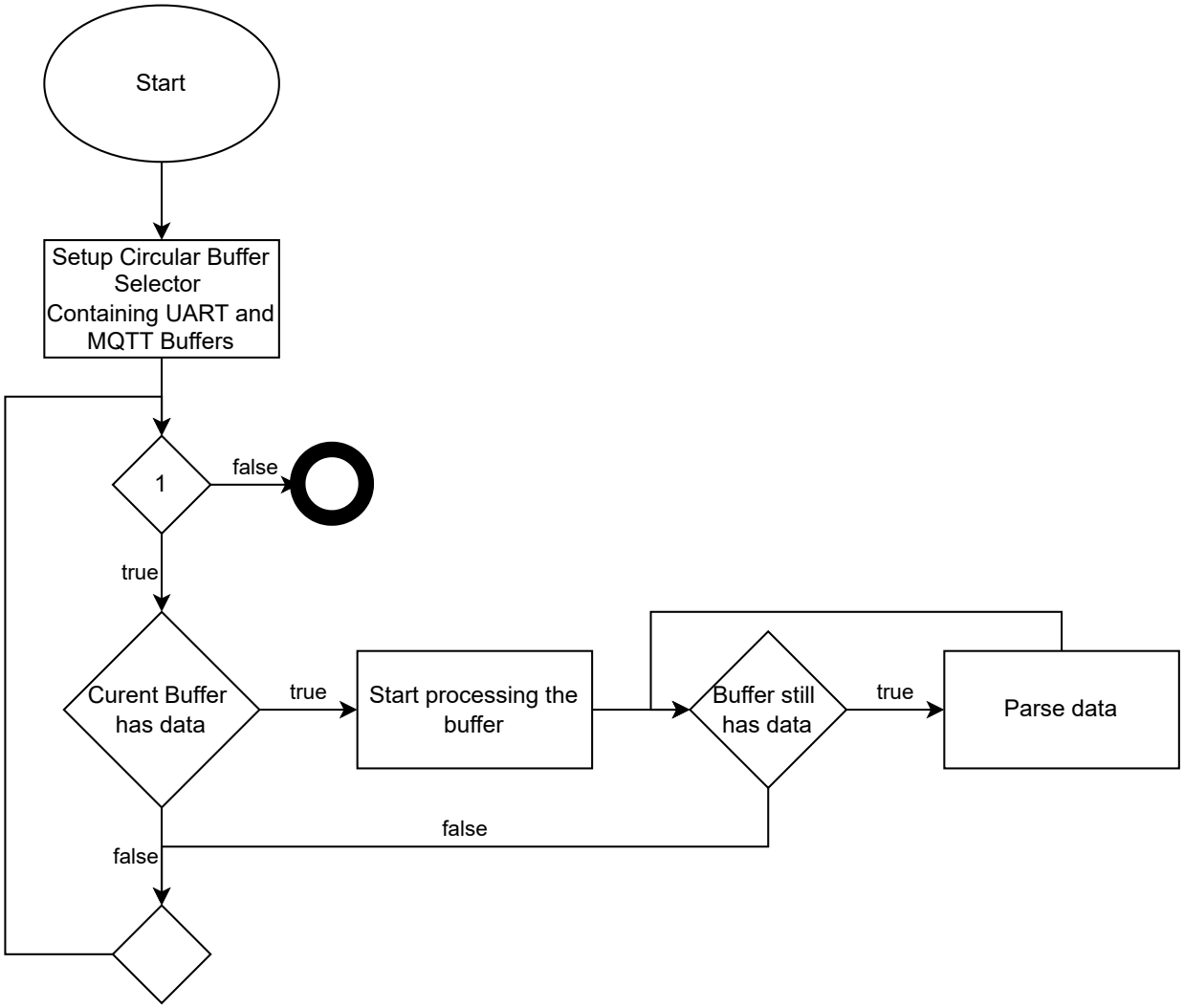
3.3 Hệ thống back-end

Hệ thống back-end là tập hợp những ứng dụng server chạy trên VPS. Những ứng dụng này bao gồm MQTT Broker, API server, và MongoDB database. Trên VPS, hệ thống back-end được triển khai bằng kỹ thuật Docker Containerizing.

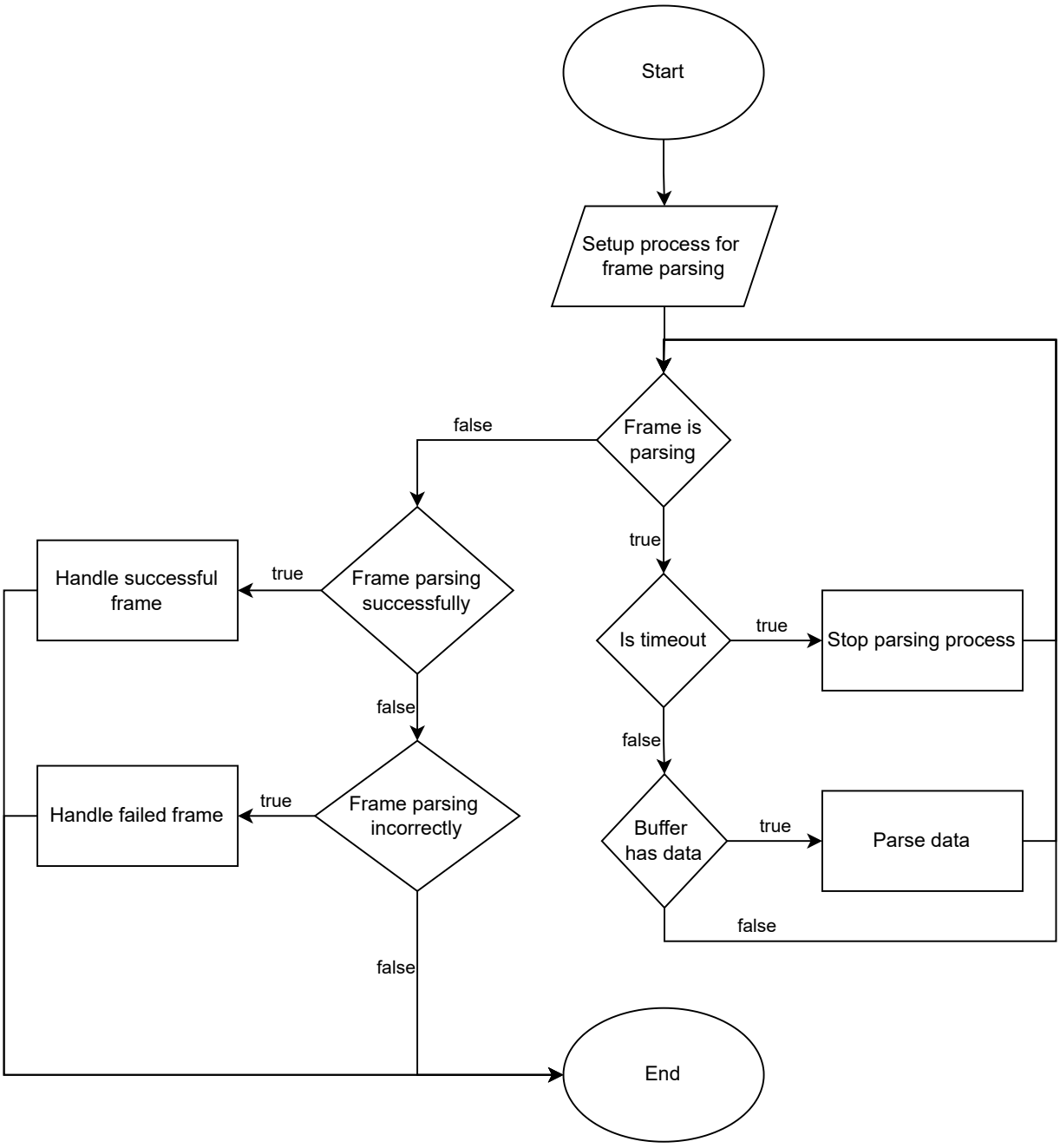
3.3.1 MQTT Broker

MQTT Broker, trung tâm của giao thức Publish/Subscribe MQTT, là một ứng dụng server thu nhận tất cả các message từ các MQTT client và định tuyến đến những subscribing client thích hợp [6].

Trong hệ thống IoT được đề xuất, MQTT Broker được xây dựng trên framework



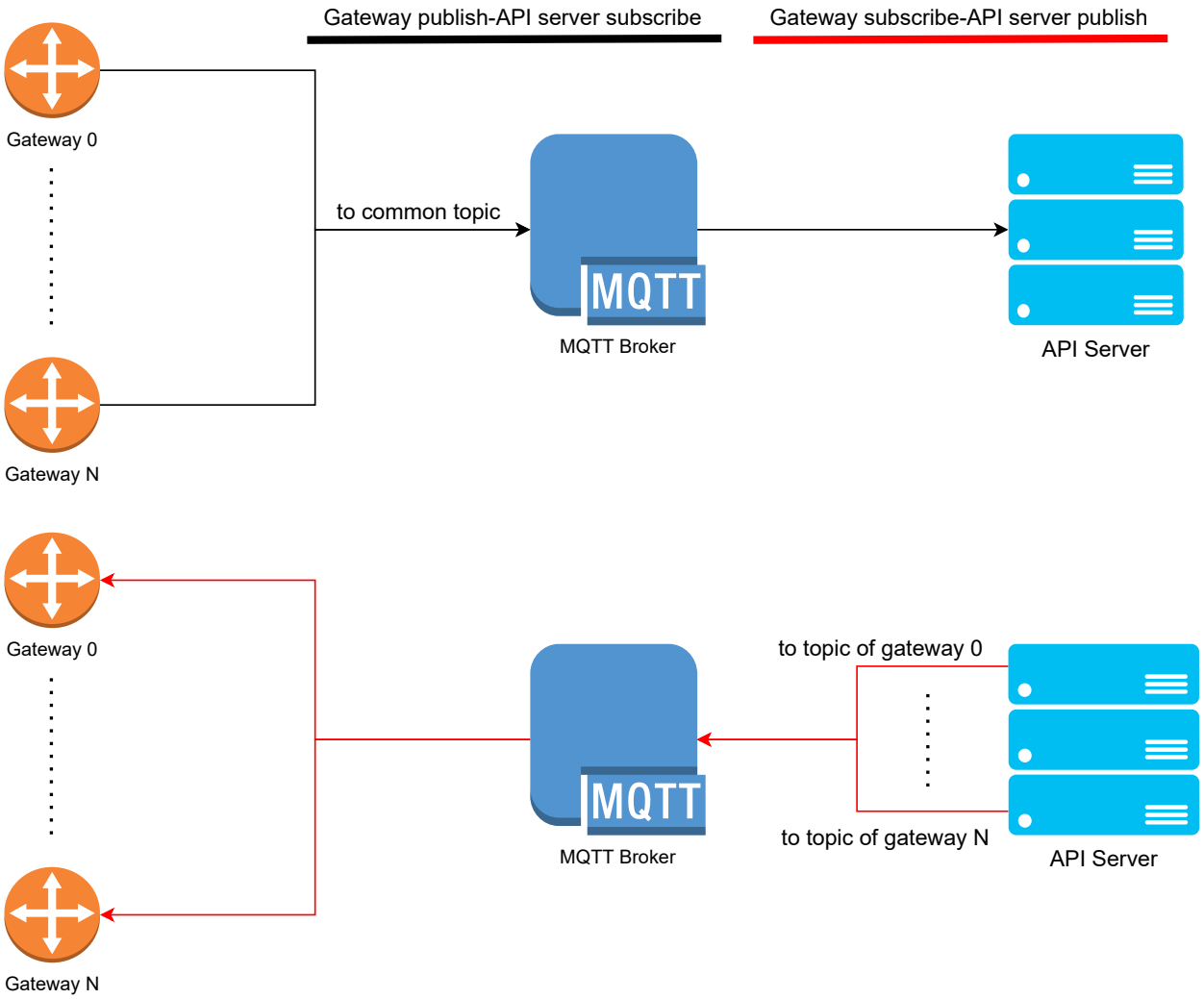
Hình 3.3: Quá trình phân xử các bộ đệm vòng.



Hình 3.4: Process phân giải frame.

NodeJS và ngôn ngữ JavaScript. Trong MQTT Broker, các MQTT client là IoT gateway và API server. Hơn nữa, các dữ liệu publish/subscribe trên MQTT Broker đều được mã hóa bởi thuật toán ChaCha20-Poly1305.

Về tổ chức kết nối, giao thức MQTT sử dụng các topic để quản lý luồng dữ liệu. Trên hệ thống IoT, các gateway và API server sẽ được tổ chức kết nối MQTT thông qua việc publish/subscribe (minh họa như hình 3.5). Trong đó, API server subscribe vào một topic và các gateway cùng gửi dữ liệu từ phần cứng vào cùng một topic đó. Trong khi, các gateway sẽ subscribe vào một topic riêng biệt và API server gửi dữ liệu tới phần cứng thông qua từng topic riêng biệt đó. Các topic của gateway được phân biệt thông qua 12-byte gateway ID.



Hình 3.5: Mô hình Publish/Subscribe MQTT trên hệ thống IoT.

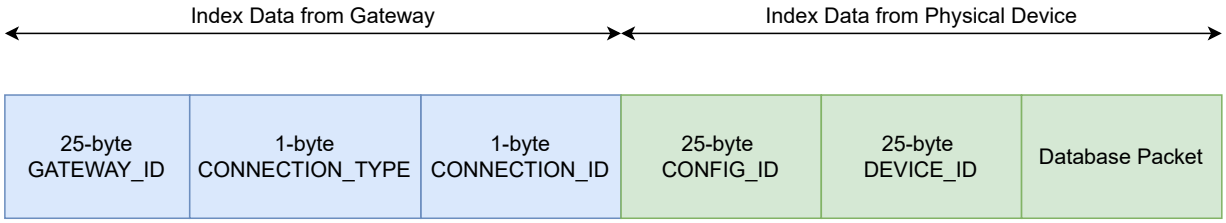
3.3.2 API Server

API server, trung tâm của hệ thống IoT, là ứng dụng server được xây dựng trên nền tảng EggJS, môi trường NodeJS, và ngôn ngữ JavaScript. Trên hệ thống, API server có nhiệm vụ thu nhận và xử lý dữ liệu cũng như cung cấp dịch vụ truy xuất database cho phần cứng và giao diện web. Cụ thể, API server thu nhận và xử lý dữ liệu phần cứng thông qua giao thức MQTT và dữ liệu từ giao diện web thông qua giao thức HTTPS. Cuối cùng, API server là một điểm kết nối với database và cung cấp các dịch vụ truy xuất và sự kiện từ database.

3.3.2.1 Giao tiếp với phần cứng

API server giao tiếp với phần cứng thông qua kết nối MQTT Broker và đóng vai trò là một MQTT client. Trên giao tiếp này, giao thức frame (được đề cập ở mục 2.4.2) được sử dụng để truyền và nhận dữ liệu từ phần cứng. Cấu trúc gói dữ liệu trên frame giao tiếp với phần cứng (minh họa như hình 3.6) bao gồm: 25-byte GATEWAY_ID, 1-byte CONNECTION_TYPE, 1-byte CONNECTION_ID, 25-byte CONFIG_ID, 25-byte DEVICE_ID, và gói dữ liệu để cập nhật database. Trong đó:

- 25-byte GATEWAY_ID là ID được cài đặt trên gaetway.
- 1-byte CONNECTION_TYPE và 1-byte CONNECTION_ID là chỉ số kết nối với thiết bị. Chỉ số này giúp gateway định tuyến dữ liệu tới đúng thiết bị vật lý đang kết nối với nó.
- 25-byte CONFIG_ID và 25-byte DEVICE_ID là ID của cấu hình và ID của thiết bị vật lý.



Hình 3.6: Cấu trúc gói dữ liệu trên frame giao tiếp với phần cứng.

3.3.2.2 Giao tiếp với giao diện web

API server đóng vai trò như một Representational state transfer (REST) API và Websocket server trong giao tiếp với giao diện web.

Trong vai trò REST API server, ứng dụng server cung cấp các dịch vụ tương tác với database theo cấu trúc Create, read, update, and delete (CRUD) và phong cách REST (minh họa như bảng 3.1). Dựa trên cấu trúc này, giao diện web có thể truy xuất database thông qua các phương thức HTTP với cấu trúc CRUD tương ứng. Hiện tại, giao diện web có thể truy xuất tới 5 REST API. Đó là:

- "config": dịch vụ truy xuất dữ liệu các cấu hình chung cho tất cả thiết bị phần cứng.
- "device": dịch vụ truy xuất dữ liệu của một thiết bị vật lý cụ thể.
- "UI": dịch vụ truy xuất dữ liệu của một UI dashboard linh hoạt.
- "user": dịch vụ truy xuất dữ liệu của một người dùng.
- "vStorage": dịch vụ truy xuất dữ liệu của một VS.

Bảng 3.1: Cấu trúc CRUD theo phong cách REST. Nguồn từ Wikipedia [7].

CRUD	**HTTP**
Create	POST, PUT
Read	GET
Update	PUT, PATCH
Delete	DELETE

Trong vai trò Websocket server, API server đồng bộ (đề cập ở mục 2.4.3) các sự thay đổi dữ liệu database với giao diện web. Cụ thể, dựa trên sự kiện "change streams" của MongoDB, dữ liệu của các VS sẽ luôn được thông báo tới giao diện web. Cấu trúc dữ liệu gửi tới giao diện web là kiểu dữ liệu JavaScript Object Notation (JSON) (minh họa như hình 3.7). Trong đó:

- Trường "cmd" là lệnh yêu cầu đồng bộ dự liệu VS.
- Trường "data" là trường dữ liệu cần đồng bộ. Trường này chứa ID của VS (**_vs_id**), device ID (**dev_id**), dữ liệu đồng bộ (**data**), và toàn bộ thuộc tính của VS (**fullDocument**).

```
type DESIoT_VSSyncPacket = {
  cmd: VSSyncCMDs;
  data: {
    _vs_id: string;
    dev_id: string;
    data: number | string | undefined;
    fullDocument: DESIoTVStorageType;
  };
};
```

Hình 3.7: Cấu trúc dữ liệu JSON đồng bộ với giao diện web.

3.3.2.3 Giao tiếp với database

API server tạo kết nối với MongoDB thông qua thư viện MongooseJS. Cụ thể, ứng dụng server kết nối với MongoDB qua phương thức **mongoose.connect()**. Cuối cùng, mongoose cung cấp giải pháp đơn giản, dựa trên schema để lập mô hình (model) dữ liệu trên ứng dụng server [8]. Các model trong giao tiếp này bao gồm:

- "config": cấu trúc document của các cấu hình chung cho tất cả thiết bị phần cứng.
- "device": cấu trúc document của một thiết bị vật lý cụ thể.
- "UI": cấu trúc document của một UI dashboard linh hoạt.
- "user": cấu trúc document của một người dùng.
- "vStorage": cấu trúc document của một VS.

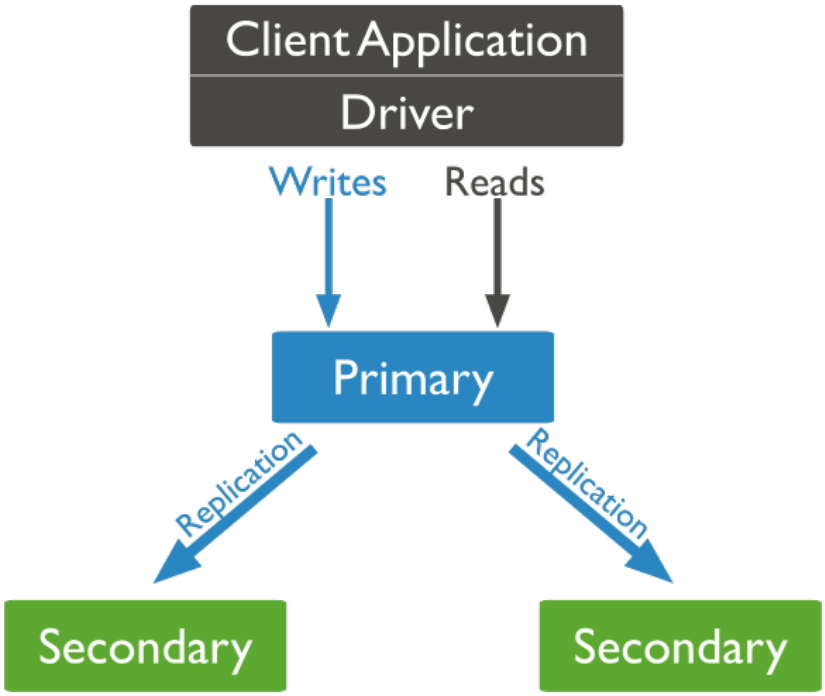
3.3.3 MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu miễn phí và mã nguồn mở đa nền tảng. Được phân loại như một hệ quản trị cơ sở dữ liệu NoSQL, MongoDB sử dụng các JSON-like document với schema [9]. Trên VPS, MongoDB hoạt động như một ứng dụng server. Cụ thể, ứng này được cài đặt bằng "mongo Docker Image" trên **docker-compose** và hoạt động trên cùng một VLAN với API server và MQTT Broker.

MongoDB thiết lập kết nối với API server thông qua thư viện "MongooseJS" và cung cấp giải pháp cho việc đồng bộ với phần cứng và giao diện web. Cụ thể, MongoDB cung cấp giải pháp đồng bộ thông qua sự kiện "change streams". Sự kiện

này giúp ứng dụng server theo dõi các thay đổi dữ liệu thời gian thực một cách dễ dàng. "Change streams" hoạt động trên "Replica set" của MongoDB.

Trên VPS, Mongo được triển khai trên mô hình "Replica set". Một *replica set* trong MongoDB là một tập hợp các *mongod* process mà duy trì cùng một data set. *Replica set* cung cấp tính dự phòng (redundancy) và có sẵn cao (high availability), và là cơ sở cho tất cả quá trình phát triển sản phẩm [10]. Trên VPS, mô hình MongoDB *Replica set* (minh họa như hình 3.8) bao gồm một *primary* node và hai *secondary* node.

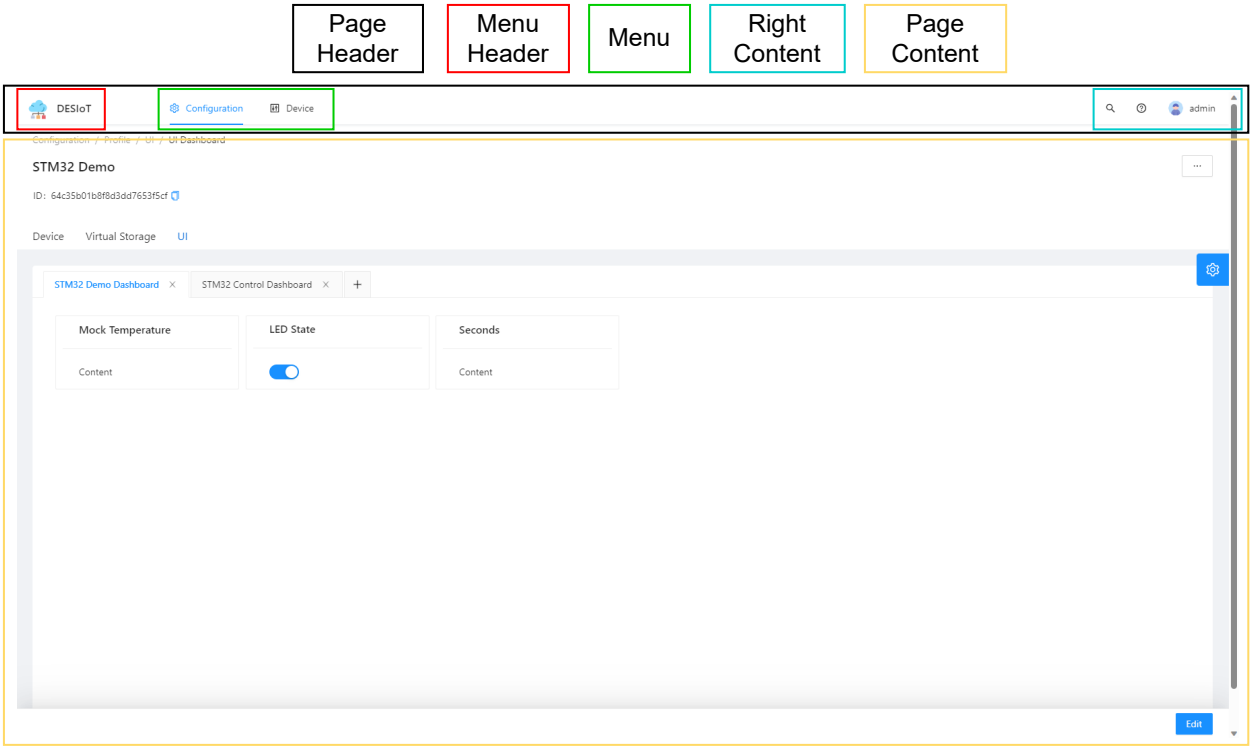


Hình 3.8: Mô hình MongoDB Replica Set trên VPS. Nguồn từ MongoDB [10].

3.4 Giao diện web

Giao diện web được xây dựng trên nền tảng Ant Design Pro, framework ReactJS, và ngôn ngữ TypeScript. Dựa trên nền tảng này, giao diện web sẽ được phát triển để cung cấp tương tác cho các admin dựa trên thư viện Ant Design và framework UmiJS. Trên nền tảng Ant Design Pro, giao diện được trang bị sẵn Main Page (minh họa như hình 3.9) và Login Page 3.10). Main Page Layout bao gồm Page Header, Menu, Menu Header, Right Content, và Page Content; và Login Page cung cấp form để đăng nhập.

Dựa trên nền tảng Ant Design Pro, giao diện web thiết lập các tương tác cho các



Hình 3.9: Layout của Main Page trên nền tảng Ant Design Pro.

quản trị viên. Các tương tác này chủ yếu diễn ra trên hai trang Configuration và Device.

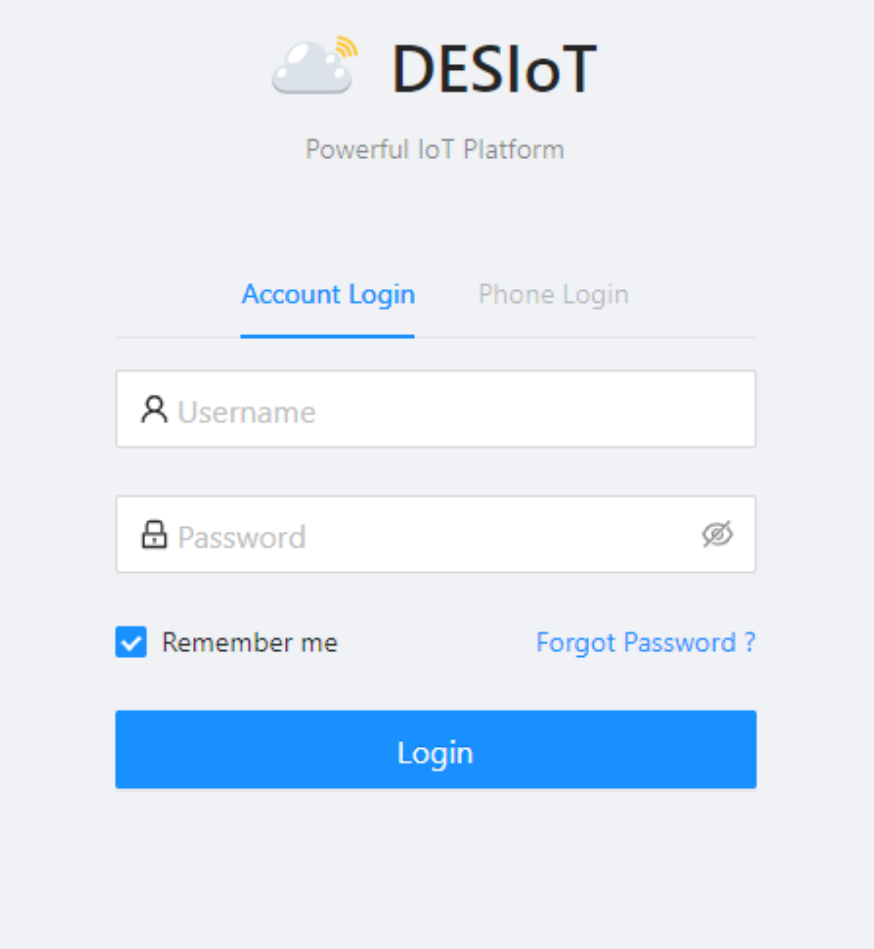
3.4.1 Configuration Page

Configuration Page (minh họa như hình 3.11) hiển thị và cung cấp tương tác cho các cấu hình trên thiết bị phần cứng. Cụ thể, trên trang này, người dùng có thể quan sát, tạo mới, và truy cập một configuration.

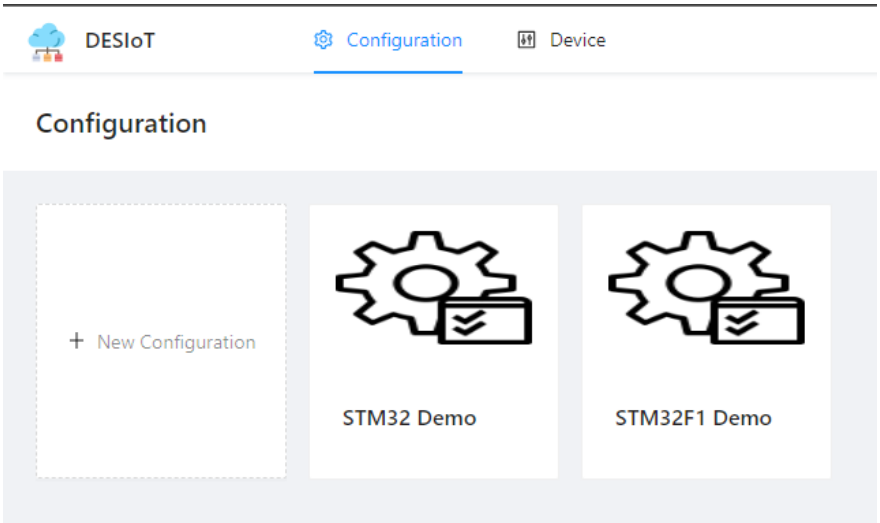
Khi mở một trang của một Configuration, người dùng có thể xem thông tin của Configuration. Ngoài ra, Configuration Page còn cung cấp các tab Device, Virtual Storage, và UI. Các tab này giúp truy xuất cơ sở dữ liệu trên model của một thiết bị phần cứng, VS, và UI dashboard.

Trên tab Device, Device Tab (minh họa như hình 3.12) hiển thị danh sách thiết bị và người dùng có thể tạo mới hoặc xóa dữ liệu của một thiết bị thông qua **Add device** hoặc **Delete** button tương ứng. Dữ liệu của một device bao gồm "Name" và "ID".

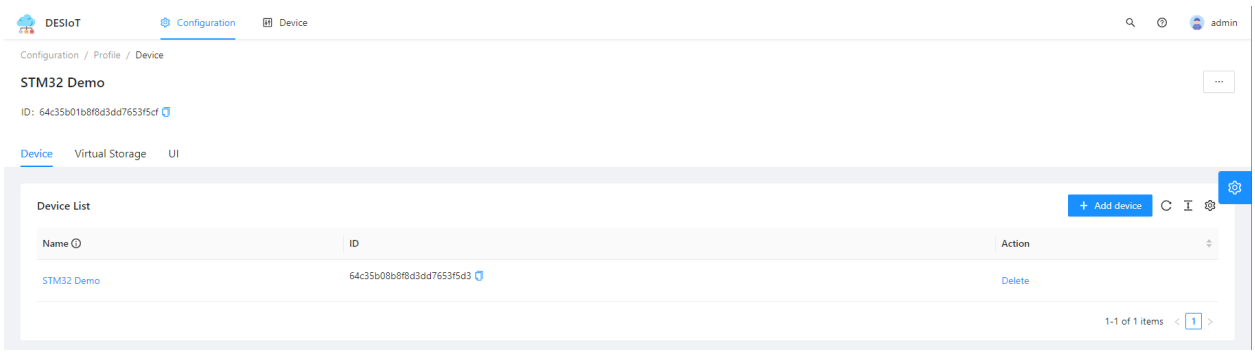
Trên tab Virtual Storage, Virtual Storage Tab (minh họa như hình 3.13) hiển thị danh sách các VS và người dùng có thể tạo mới hoặc xóa dữ liệu của một VS thông qua **Add Virtual Storage** hoặc **Delete** button tương ứng. Dữ liệu của một VS bao



Hình 3.10: Login Page trên nền tảng Ant Design Pro.

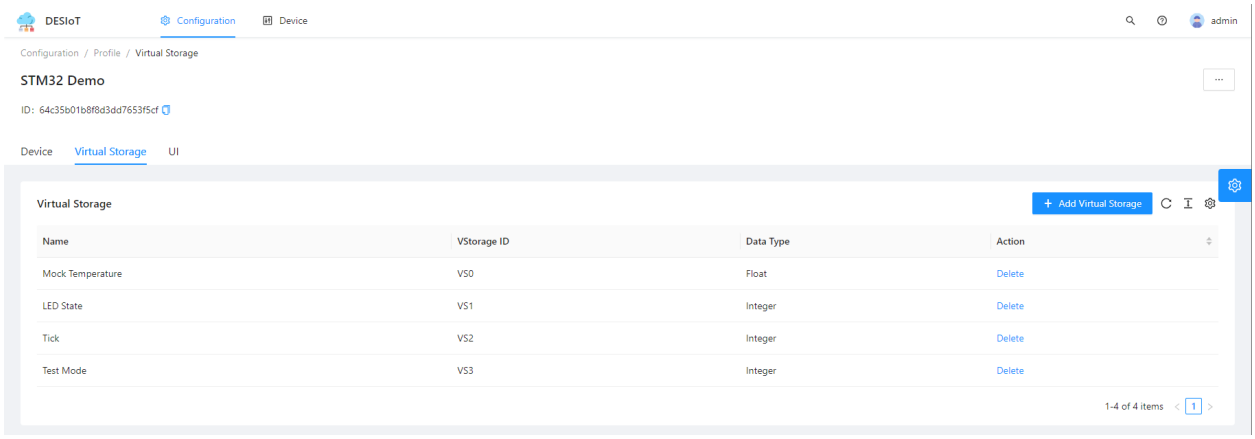


Hình 3.11: Giao diện của Configuration Page.



Hình 3.12: Giao diện của Device Tab trên Configuration Page.

gồm "Name", "VStorage ID", và "Data Type".

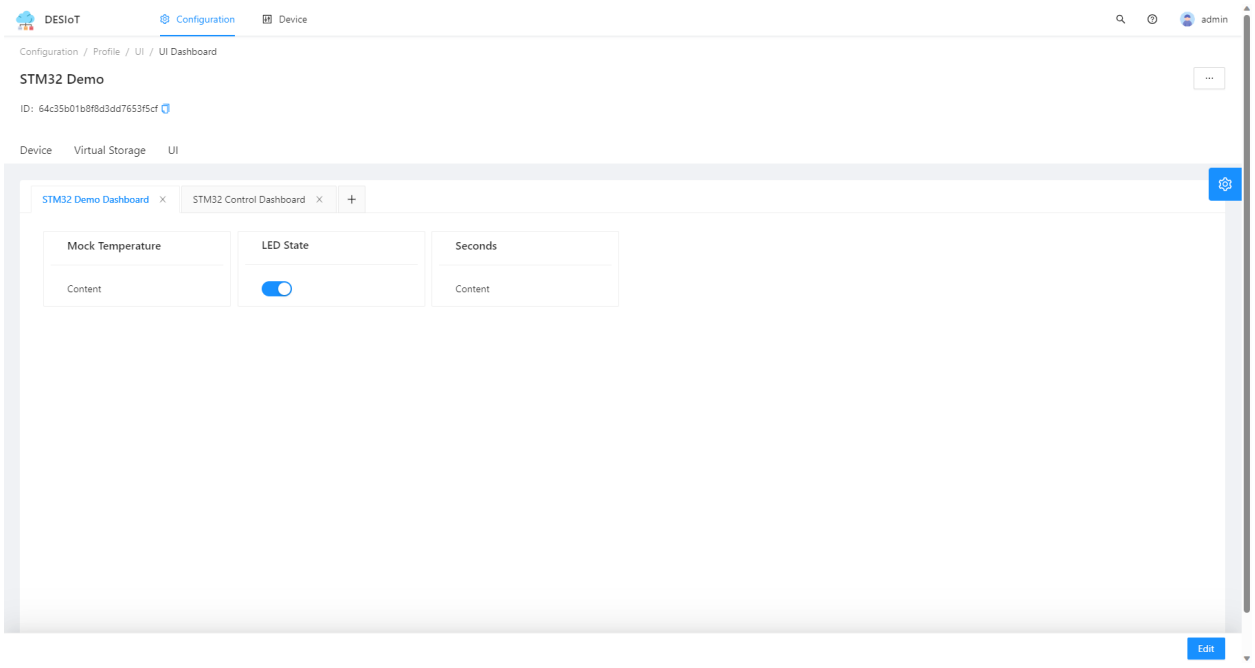


Hình 3.13: Giao diện của Virtual Storage Tab trên Configuration Page.

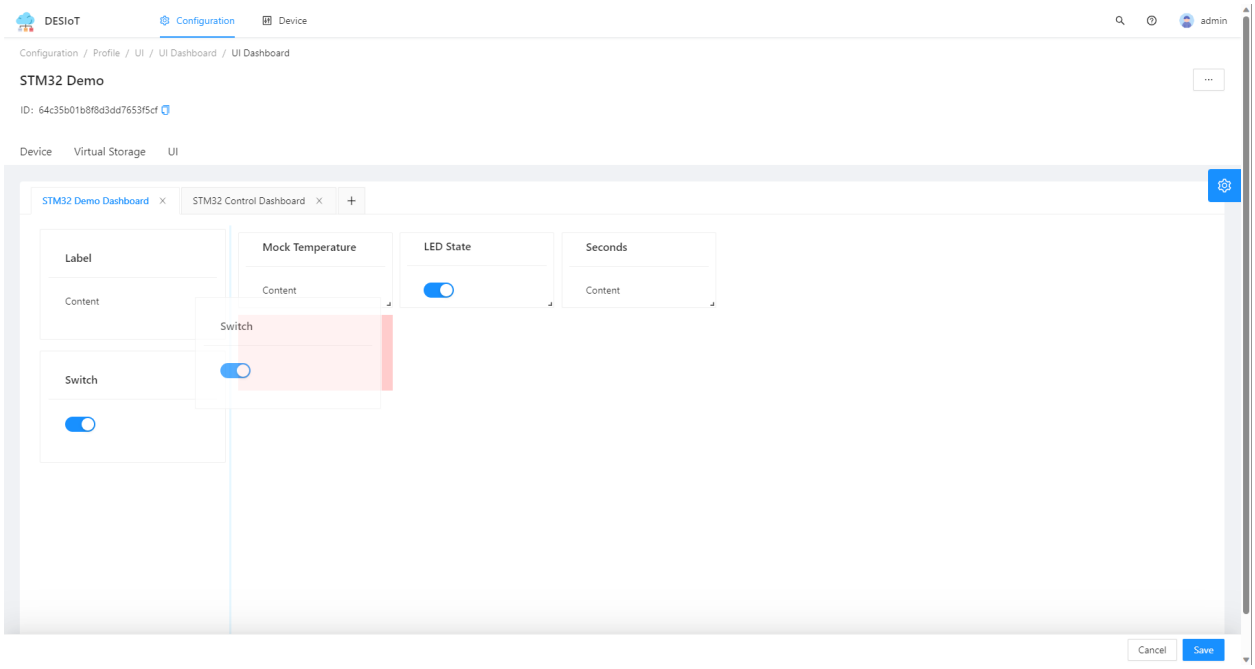
Trên tab UI, UI Tab (minh họa như hình 3.14) hiển thị danh sách các Dashboard tab và người dùng có thể tạo mới hoặc xóa dữ liệu của một Dashboard thông qua "+" hoặc "X" button tương ứng. Trên mỗi Dashboard, người dùng có thể bật chế độ chỉnh sửa (minh họa như hình 3.15) thông qua **Edit** button. Trên chế độ này, người dùng có thể thêm, xóa, hoặc chỉnh sửa các widget có sẵn để thiết kế Dashboard.

3.4.2 Device Page

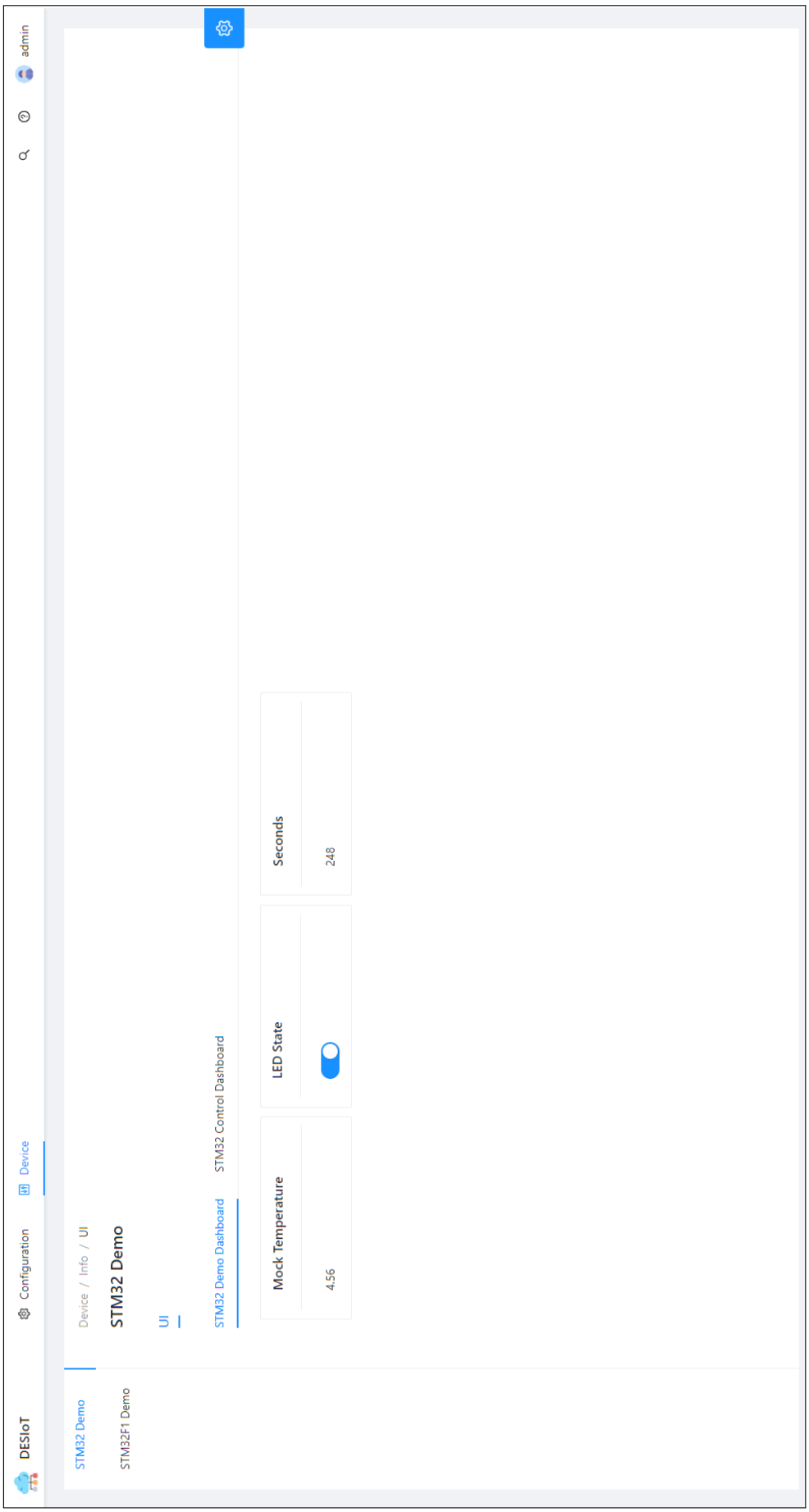
Device Page (minh họa như hình 3.16) hiển thị danh sách các thiết bị đã cấu hình trên Configuration Page. Khi mở một Device Tab, người dùng có thể lựa chọn UI Dashboard đã tạo và quan sát dữ liệu thời gian thực và điều khiển thiết bị trên dashboard.



Hình 3.14: Giao diện của UI Tab trên Configuration Page.



Hình 3.15: Giao diện của UI Tab chế độ Edit trên Configuration Page.



Hình 3.16: Giao diện của Device Page.

CHƯƠNG 4: KẾT QUẢ

Chương 4 đưa ra kết quả về việc hoạt động và các kỹ thuật triển khai trên phần cứng, VPS, và giao diện web.

4.1 Kết quả trên phần cứng

Kết quả trên phần cứng được trình bày, bao gồm kết quả hoạt động của kỹ thuật giao tiếp, đồng bộ, và mật mã hóa.

4.1.1 Kỹ thuật giao tiếp

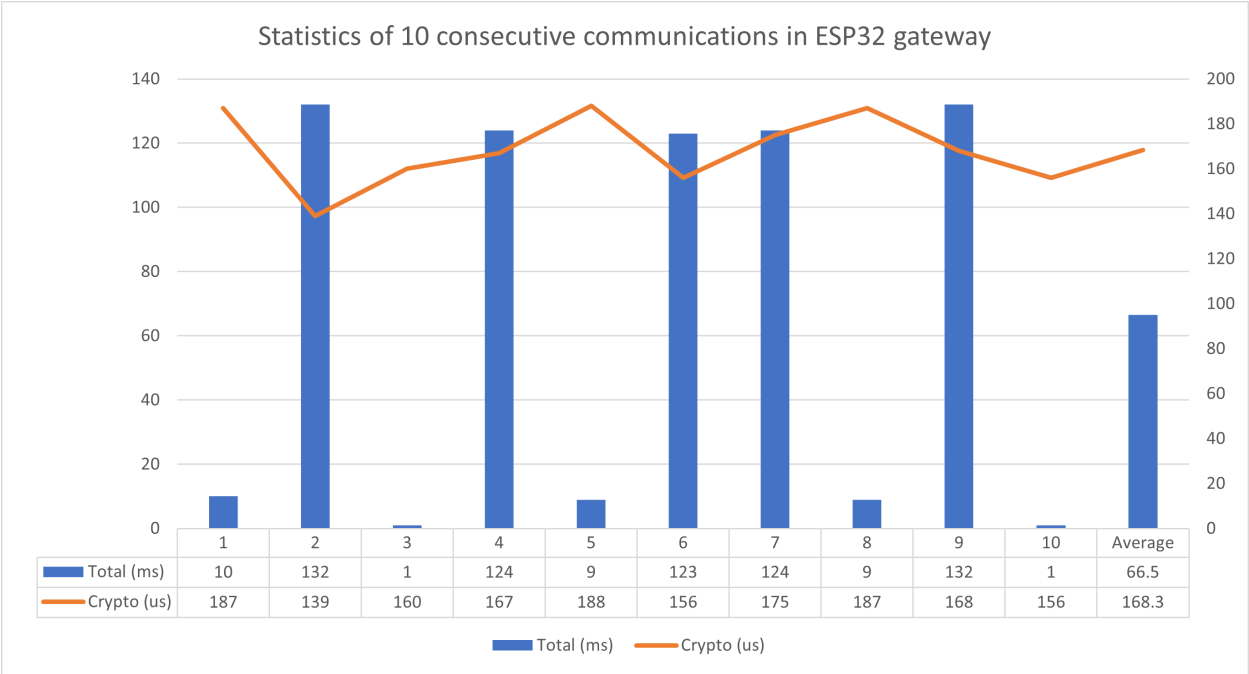
Kỹ thuật giao tiếp được cài đặt như nhau trên MCU STM32 và ESP32. Vì vậy, dựa trên log của MCU ESP32, một giao tiếp với device và Application programming interface (API) server điển hình (minh họa như hình 4.1) sẽ diễn ra trong khoảng $100ms \rightarrow 150ms$. Thống kê của 10 lần giao tiếp liên tiếp (minh họa như hình 4.2) cho thấy quá trình giao tiếp diễn ra trong khoảng $66.5ms$ và mã hóa/giải mã diễn ra trong $167\mu s$.

```
- Device Communication Start
        - H1 OK
        - H2 OK
        - T1 OK
        - T2 OK
        - CRC OK
        - Encrypt time 179 us
- Device Communication End (137 ms), data length = 98 bytes
- Server Communication Start
        - H1 OK
        - H2 OK
        - T1 OK
        - T2 OK
        - CRC OK
        - DECRYPT SUCCESSFUL (159 us)
- Server Communication End (124 ms), data length = 5 bytes
```

Hình 4.1: Một giao tiếp điển hình với device và server.

4.1.2 Kỹ thuật mật mã hóa

Kỹ thuật mật mã hóa sử dụng thuật toán AEAD ChaCha20-Poly1305 được triển khai thành công trên MCU ESP32. Theo kiến trúc của thuật toán AEAD, quá trình



	1	2	3	4	5	6	7	8	9	10	Average
Total (ms)	10	132	1	124	9	123	124	9	132	1	66.5
Crypto (us)	187	139	160	167	188	156	175	187	168	156	168.3

Hình 4.2: Thống kê của 10 quá trình giao tiếp liên tiếp.

mã hóa (minh họa như hình 4.3) đã diễn ra đúng đắn và tuần tự theo các quá trình của RFC 7539 [5]:

1. Thiết lập State cho Poly1305 one-time key.
2. Tạo Poly1305 one-time key.
3. Thực thi ChaCha20 encryption function.
4. Thực thi Poly1305 function cho ciphertext.

Và quá trình giải mã (minh họa như hình 4.4) đã diễn ra đúng đắn và tuần tự theo các quá trình của RFC 7539 [5]:

1. Thiết lập State cho Poly1305 one-time key.
2. Tạo Poly1305 one-time key.
3. Thực thi Poly1305 function cho ciphertext.
4. Thực thi ChaCha20 dencryption function.

```
- ChaCha20-Poly1305 Encryption Start
        - Set-up of Poly1305 one-time key:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000000 43860c18 0e1c9021 20418307

        - Poly1305 one-time key:
                3d8653f0 ebba5876 5feca469 3f911b29
                6444430e 31ed471d 27a666c3 027c1891

        - ChaCha20 Encryption State:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000001 43860c18 0e1c9021 20418307

        - Poly1305 function execution for ciphertext.
        - Poly1305 Tag:
                792f389c 2aaf5ddf 9c4310a0 ab9507a0

- ChaCha20-Poly1305 Encryption End
```

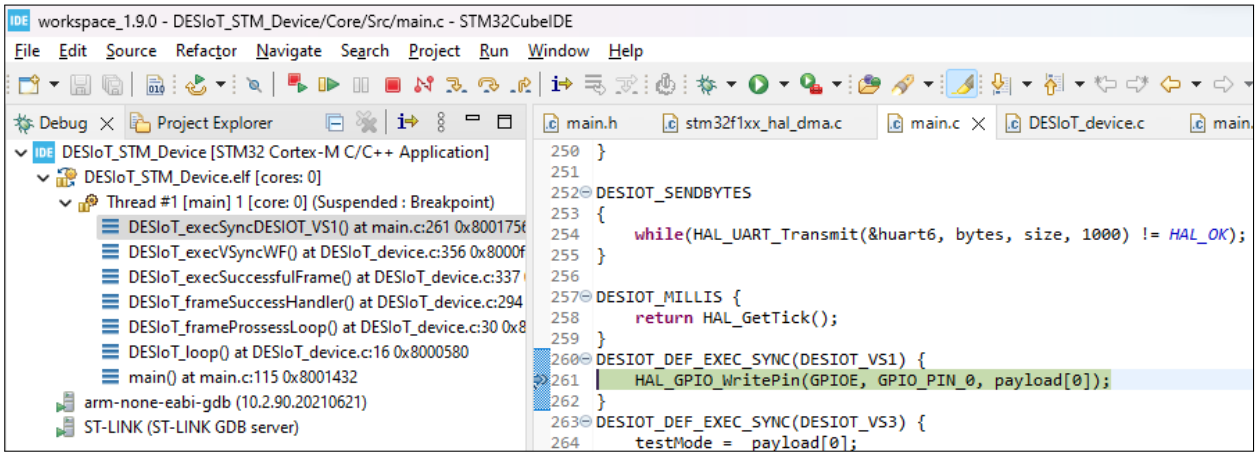
Hình 4.3: Quá trình mã hóa của thuật toán AEAD ChaCha20-Poly1305 trên MCU ESP32.

```
- ChaCha20-Poly1305 Decryption Start
        - Set-up of Poly1305 one-time key:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000000 43860c18 0e1c9021 20418307

        - Poly1305 one-time key:
                3d8653f0 ebba5876 5feca469 3f911b29
                6444430e 31ed471d 27a666c3 027c1891

        - Poly1305 function execution for ciphertext.
        - ChaCha20 Decryption State:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000001 43860c18 0e1c9021 20418307

        - Poly1305 Tag:
                98bbca26 04d8cf07 02d4f4d8 a942199a

- DECRYPT SUCCESSFUL (741565 us)
- ChaCha20-Poly1305 Decryption End
```

Hình 4.4: Quá trình giải mã của thuật toán AEAD ChaCha20-Poly1305 trên MCU ESP32.

4.1.3 Kỹ thuật đồng bộ

Kỹ thuật đồng bộ được triển khai thành công trên firmware của MCU STM32. Trên công cụ STM32CubeIDE, callback đồng bộ với VS1 (minh họa như hình 4.5) được thực hiện khi có sự thay đổi dữ liệu của VS này trên database của VPS. Callback này được theo dõi thông qua breakpoint trên IDE.



Hình 4.5: Breakpoint theo dõi callback đồng bộ với VS1 được thực hiện trên STM32CubeIDE.

4.2 Kết quả trên VPS

Trên VPS, các kết quả được minh họa bao gồm kết quả của kỹ thuật Docker Containerizing, Nginx web server, và các kết quả của giao tiếp và đồng bộ trên API server.

4.2.1 Kết quả của kỹ thuật Docker Containerizing

Hệ thống back-end bao gồm MQTT Broker, API server, và MongoDB database, đã được triển khai thành công trên VPS thông qua kỹ thuật Docker Containerizing. Cụ thể, commandline của VPS khi chạy lệnh "docker ps" (minh họa như hình 4.6) hiển thị lần lượt các container của API server, replica set của MongoDB, và MQTT Broker. Trong đó:

- API server chạy trên container **desiot-server-desiot-server-1** và hoạt động ở port 7001.
- Replica set của MongoDB chạy trên các container **mongo1**, **mongo2**, và **mongo3** cũng như hoạt động ở các port 30001 → 30003.

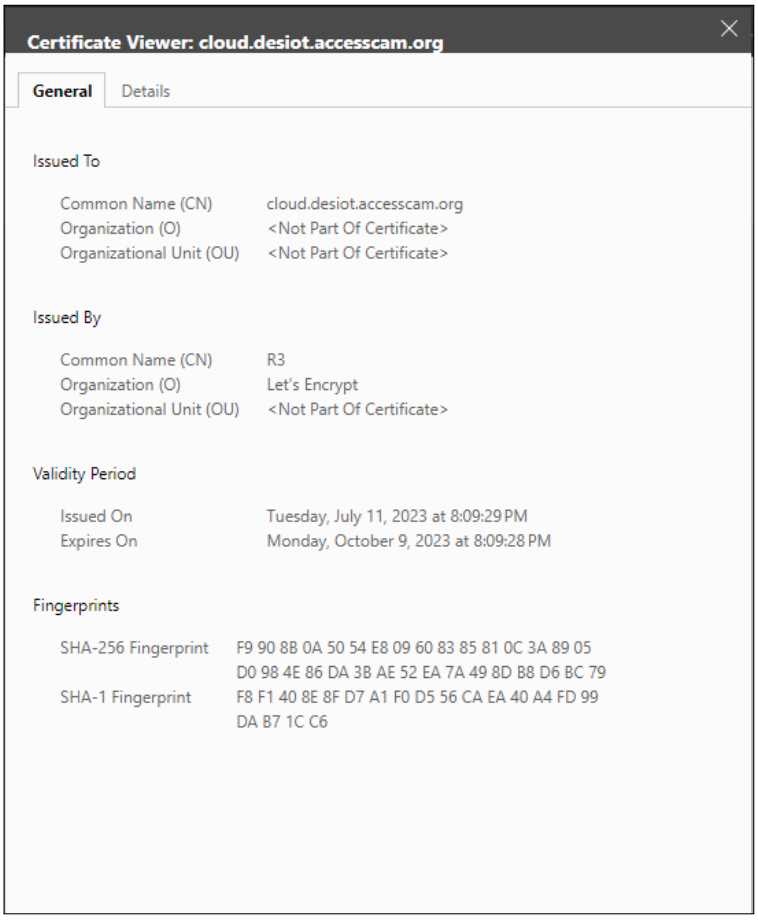
- MQTT Broker chạy trên container **iot-services-broker-1** và hoạt động ở port 1883.

```
iot-bts2@iot-bts2:~$ docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}'
NAMES                          STATUS                  PORTS
desiot-server-desiot-server-1  Up About an hour        0.0.0.0:7001→7001/tcp, :::7001→7001/tcp
mongo2                         Up 4 weeks (healthy)    27017/tcp, 0.0.0.0:30002→30002/tcp, :::30002→30002/tcp
mongo3                         Up 4 weeks (healthy)    27017/tcp, 0.0.0.0:30003→30003/tcp, :::30003→30003/tcp
mongo1                         Up 4 weeks (healthy)    27017/tcp, 0.0.0.0:30001→30001/tcp, :::30001→30001/tcp
iot-services-broker-1          Up 4 weeks              0.0.0.0:1883→1883/tcp, :::1883→1883/tcp
```

Hình 4.6: Hệ thống back-end được triển khai trên VPS.

4.2.2 Kết quả Nginx web server

Nginx web server đã host thành công giao diện web của hệ thống IoT. Cụ thể, giao diện web được host tại địa chỉ https://cloud.desiot.accesscam.org và được cấp chứng chỉ SSL (minh họa như hình 4.7).



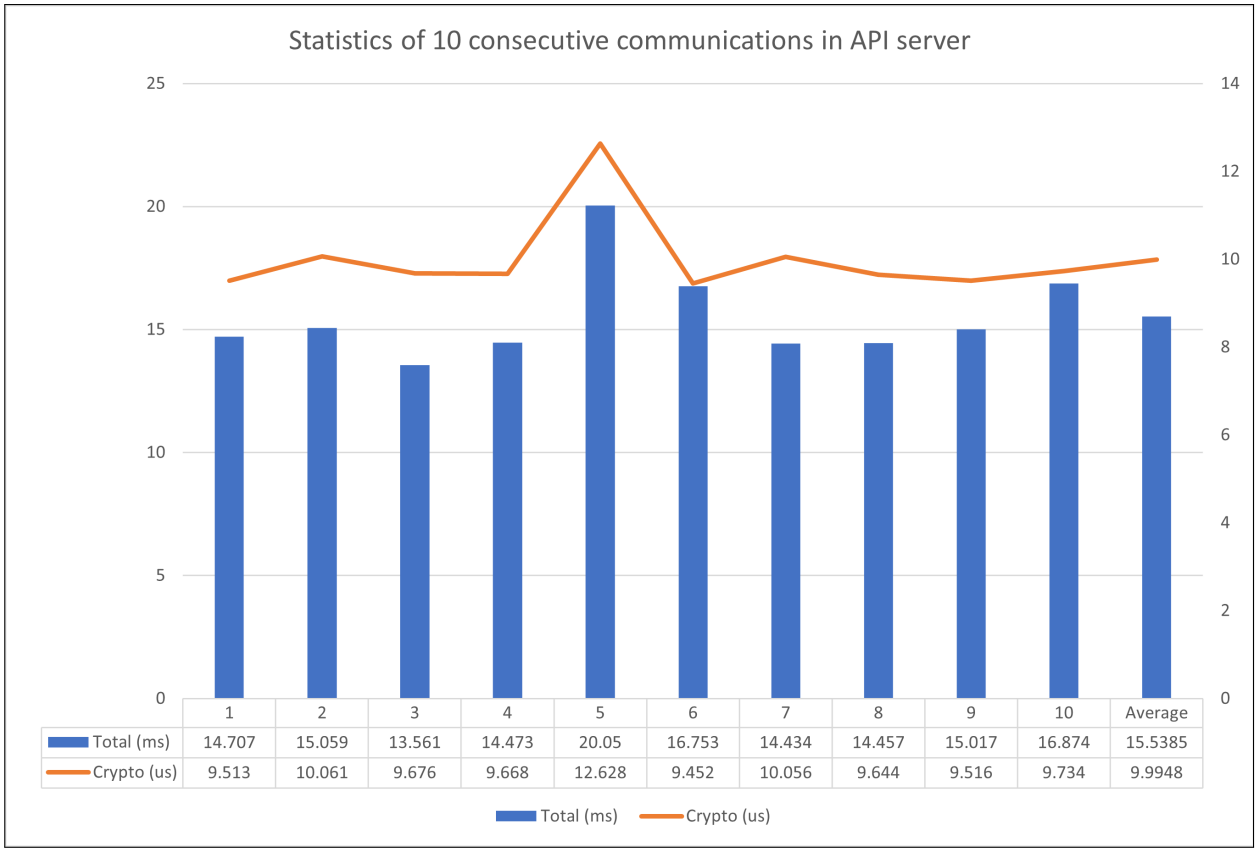
Hình 4.7: Chứng chỉ SSL của giao diện web.

4.2.3 Kết quả giao tiếp trên API server

Trên API server, quá trình phân giải và xử lý frame diễn ra khác với quá trình trên phần cứng. Cụ thể, trên API server, quá trình này diễn ra trên một frame hoàn chỉnh thay vì trên từng byte rời rạc như quá trình trên các MCU. Vì vậy, một quá trình phân giải và xử lý frame cụ thể trên API server (minh họa như hình 4.8) sẽ xử lý đồng thời việc kiểm tra các header, trailer, và CRC. Sau đó, quá trình giải mã diễn ra. Thống kê của 10 lần giao tiếp liên tiếp với gateway (minh họa như hình 4.9) có thời lượng giao tiếp trung bình với gateway diễn ra trong 15.5385ms cũng như thời gian giải mã trung bình là 9.9948μs. Thời gian trung bình giao tiếp này nhanh hơn 4.2 lần so với gateway, và thời gian giải mã trung bình nhanh hơn 16.7 lần.

```
- Communication Start, data length = 98 bytes
        - HEADERS OK
        - TRAILERS OK
        - CRC OK
        - DECRYPT time: 9.644us
[4378282.069660187] - Communication End: 14.457ms
```

Hình 4.8: Quá trình phân giải và xử lý cụ thể của một frame trên API server.



	1	2	3	4	5	6	7	8	9	10	Average
Total (ms)	14.707	15.059	13.561	14.473	20.05	16.753	14.434	14.457	15.017	16.874	15.5385
Crypto (us)	9.513	10.061	9.676	9.668	12.628	9.452	10.056	9.644	9.516	9.734	9.9948

Hình 4.9: Thống kê của 10 quá trình giao tiếp liên tiếp với gateway trên API server.

4.2.4 Kết quả kỹ thuật mật mã hóa trên API server

Kỹ thuật mật mã hóa sử dụng thuật toán AEAD ChaCha20-Poly1305 được triển khai thành công trên API server. Theo kiến trúc của thuật toán AEAD, quá trình mã hóa (minh họa như hình 4.10) đã diễn ra đúng đắn và tuần tự theo các quá trình của RFC 7539 [5]:

1. Thiết lập State cho Poly1305 one-time key.
2. Tạo Poly1305 one-time key.
3. Thực thi ChaCha20 encryption function.
4. Thực thi Poly1305 function cho ciphertext.

Và quá trình giải mã (minh họa như hình 4.11) đã diễn ra đúng đắn và tuần tự theo các quá trình của RFC 7539 [5]:

1. Thiết lập State cho Poly1305 one-time key.
2. Tạo Poly1305 one-time key.
3. Thực thi Poly1305 function cho ciphertext.
4. Thực thi ChaCha20 dencryption function.

4.2.5 Kết quả đồng bộ trên API server

Kỹ thuật đồng bộ được triển khai thành công trên API server. Trên công cụ Visual Studio Code, callback đồng bộ với model VS (minh họa như hình 4.12) được thực hiện khi có sự thay đổi dữ liệu của VS trên MongoDB database. Callback này được theo dõi thông qua breakpoint trên JavaScript Debug Terminal.

4.3 Kết quả giao diện web

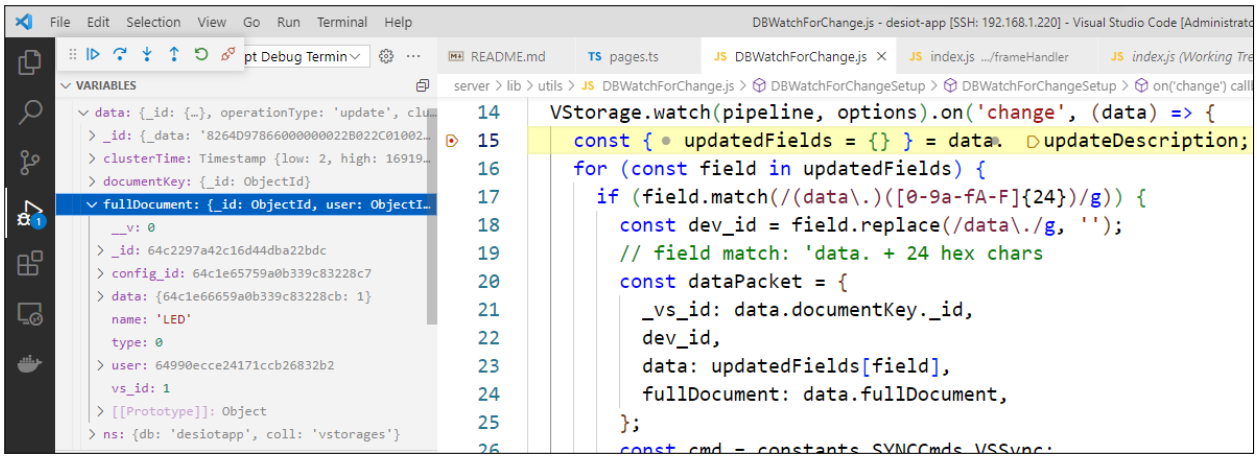
Kết quả của giao diện web minh họa các kết nối với API server, việc trực quan hóa dữ liệu của một thiết bị, và cấu hình Progressive web application (PWA).

```
- ChaCha20-Poly1305 Encryption start
        - Set-up of Poly1305 one-time key:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000000 43860c18 0e1c9021 20418307
        - ChaCha20 Encryption State:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000001 43860c18 0e1c9021 20418307
        - Poly1305 function execution for ciphertext.
        - Poly1305 Tag:
                98bbca26 04d8cf07 02d4f4d8 a942199a
- ChaCha20-Poly1305 process end
```

Hình 4.10: Quá trình mã hóa của thuật toán AEAD ChaCha20-Poly1305 trên API server.

```
- ChaCha20-Poly1305 Decryption start
        - Set-up of Poly1305 one-time key:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000000 43860c18 0e1c9021 20418307
        - Poly1305 function execution for ciphertext.
        - ChaCha20 Decryption State:
                61707865 3320646e 79622d32 6b206574
                3c78f0e1 f0e10f1e 0f1e3c78 3c78f0e1
                53a60f1e 254a9429 c993264c 5fbf3264
                00000001 43860c18 0e1c9021 20418307
        - Poly1305 Tag:
                792f389c 2aaf5ddf 9c4310a0 ab9507a0
- ChaCha20-Poly1305 process end
- Tag matched!
```

Hình 4.11: Quá trình giải mã của thuật toán AEAD ChaCha20-Poly1305 trên API server.



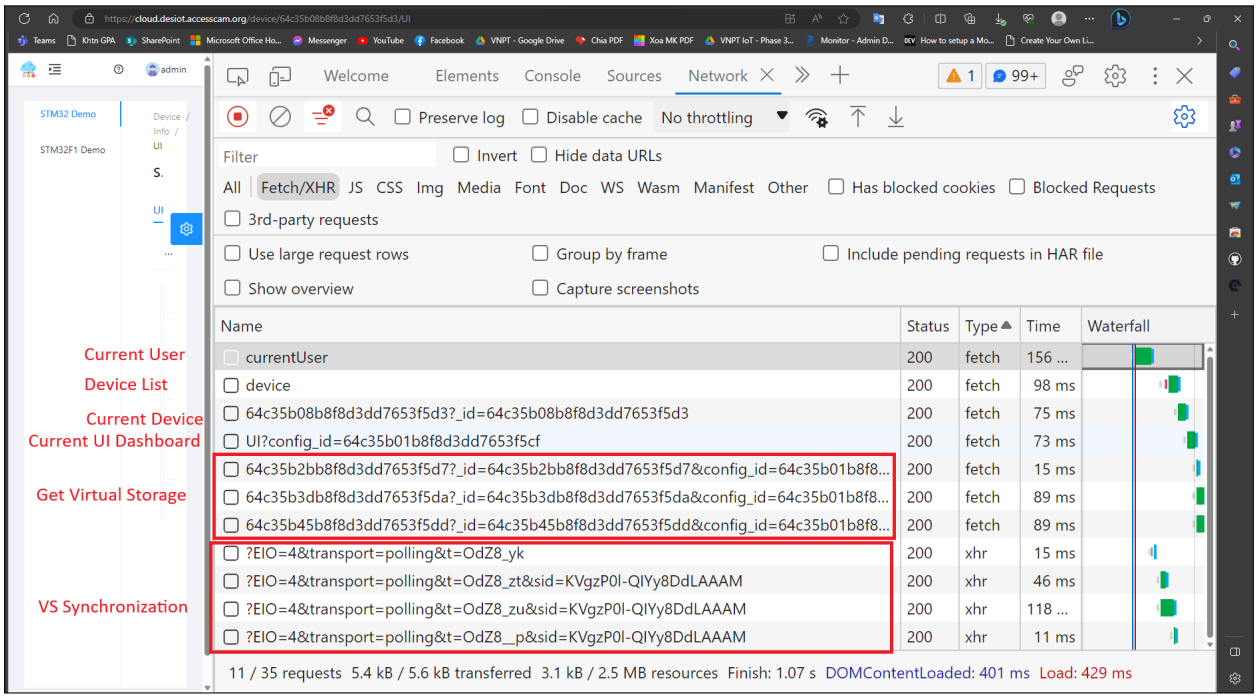
Hình 4.12: Breakpoint theo dõi callback đồng bộ với VS được thực hiện trên Visual Studio Code.

4.3.1 Kết nối với API server

Giao diện web đã tổ chức thành công các kết nối với API server. Cụ thể, khi truy cập vào giao diện của một thiết bị vật lý, network record của giao diện (minh họa như hình 4.13) ghi lại các kết nối HTTPS để lấy các dữ liệu khởi tạo và các kết nối Websocket để đồng bộ với sự thay đổi của các VS. Trong đó, các kết nối HTTPS (kiểu fetch) bao gồm:

- **Current User**: lấy dữ liệu của người dùng hiện tại.
- **Device List**: lấy danh sách dữ liệu của tất cả thiết bị.
- **Current Device**: lấy dữ liệu của một thiết bị.
- **Current UI Dashboard**: lấy dữ liệu của một UI tab.
- **Get Virtual Storage**: lấy danh sách dữ liệu của tất cả VS.

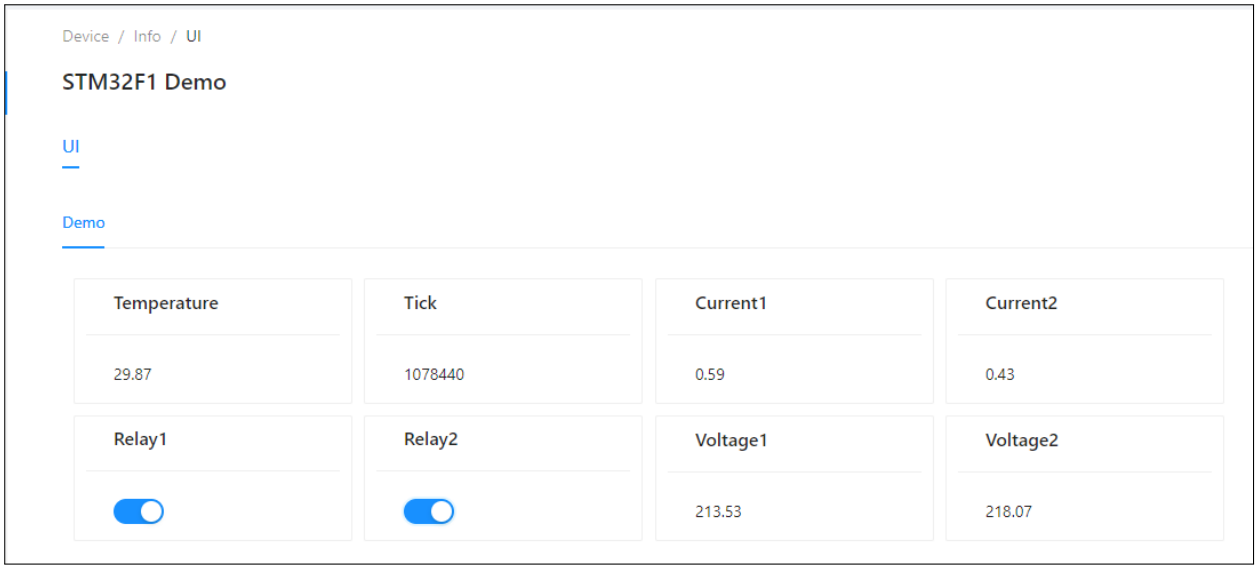
Và các kết nối Websocket (kiểu xhr) - **VS Synchronization** bao gồm các request yêu cầu đồng bộ với sự kiện thay đổi dữ liệu các VS trên API server.



Hình 4.13: Các network record hiển thị các kết nối giao diện web và API server.

4.3.2 Trực quan hóa dữ liệu của thiết bị

Trong việc trực quan hóa dữ liệu của một thiết bị, giao diện web đã hiển thị được các giá trị của database và hiển thị theo thay đổi thời gian thực. Cụ thể, giao diện web trực quan hóa dữ liệu phần cứng (minh họa như hình 4.14), đã hiển thị dữ liệu phần cứng trên database thông qua các widget trên UI dashboard.

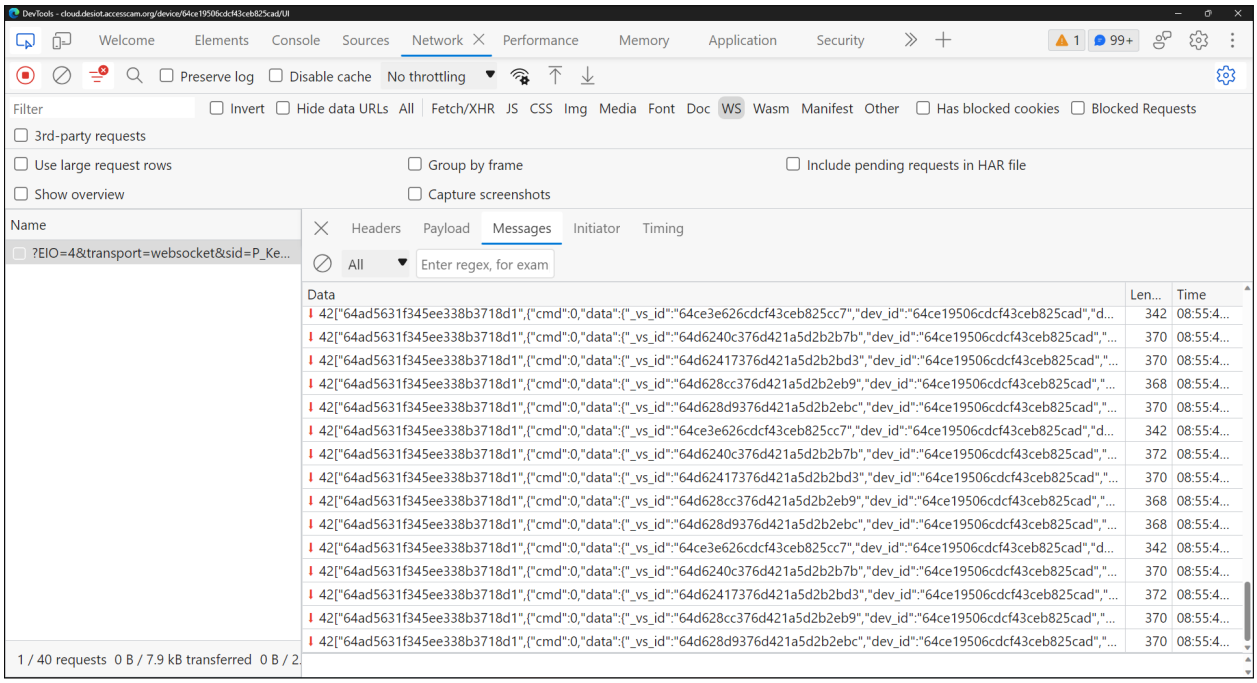


Hình 4.14: Giao diện web trực quan hóa dữ liệu phần cứng.

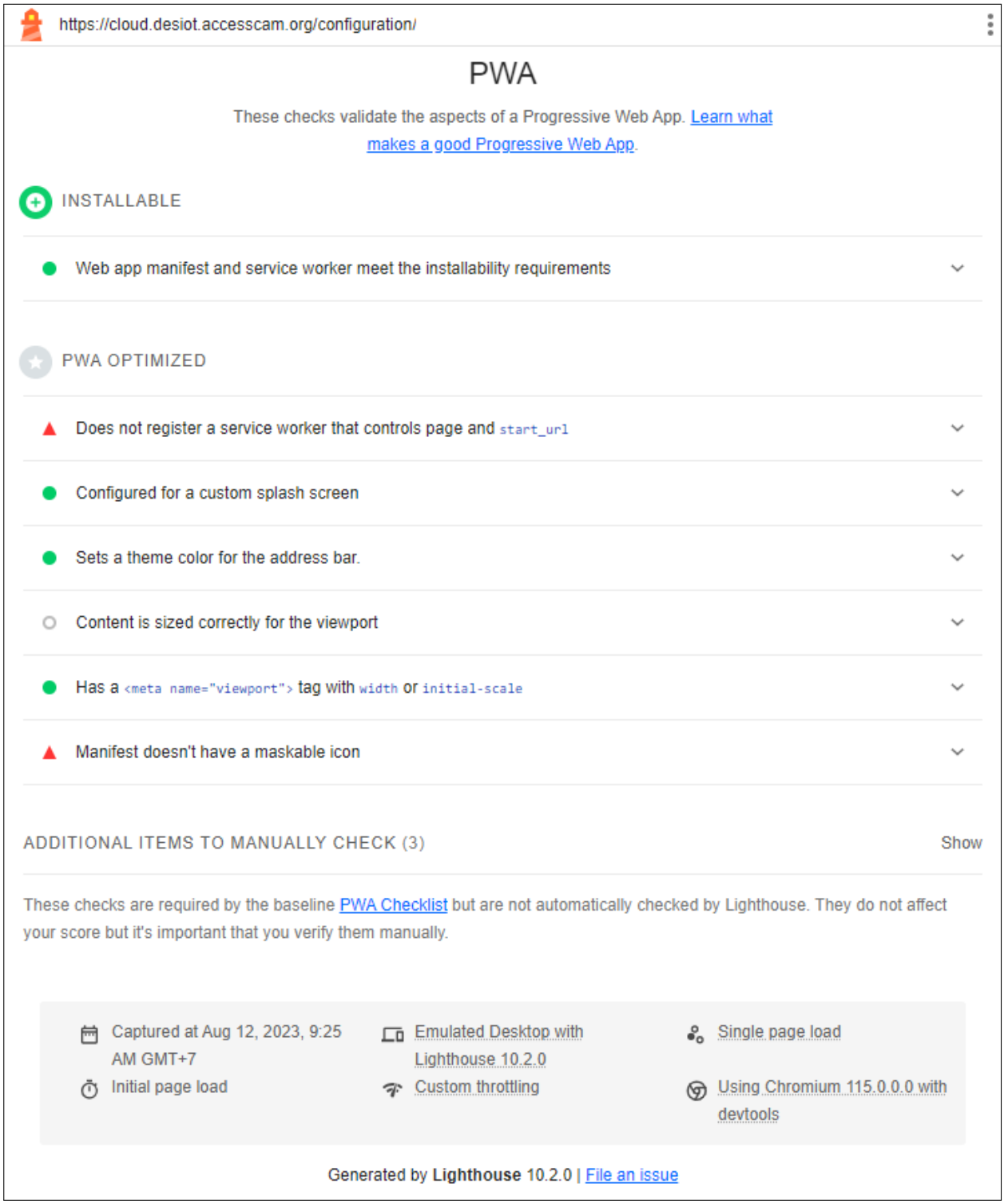
Về việc hiển thị dữ liệu thời gian thực, các network record Websocket của giao diện web (minh họa như hình 4.15) cho thấy dữ liệu phần cứng trên database luôn được đồng bộ real-time với các thành phần trên UI dashboard. Các record này được ghi lại thông qua Microsoft Edge Developer Tools.

4.3.3 Cấu hình PWA

Cấu hình PWA đã được triển khai trên giao diện web. Do đó, người dùng có thể cài đặt trang web như một ứng dụng trên nhiều nền tảng (desktop, mobile,...). Cụ thể, Lighthouse report (minh họa như hình 4.16) đã chứng thực cấu hình PWA của giao diện web.



Hình 4.15: Các Websocket record trên giao diện web ghi lại thông qua Microsoft Edge Developer Tools.



Hình 4.16: Lighthouse report của cấu hình PWA của giao diện web.

CHƯƠNG 5: KẾT LUẬN

Chương 5 mô tả ngắn gọn về mục tiêu, phương pháp, kết quả đạt được, ý nghĩa của kết quả cũng như những hạn chế, giới hạn của đề tài và hướng phát triển.

Hệ thống IoT hướng tới việc sử dụng bo mạch MCU phù hợp với việc giám sát BTS. Theo đó, cơ chế giao tiếp và đồng bộ được thiết kế để hệ thống back-end và phần cứng tương tác với nhau một cách đáng tin cậy và có khả năng tự phục hồi. Về giao diện, giao diện web linh hoạt sẽ được xây dựng với khả năng tùy chỉnh các thành phần hiển thị. Trên cơ sở giao tiếp, kỹ thuật mật mã hóa nhẹ sẽ được triển khai để đảm bảo tính bảo mật trong các giao tiếp của hệ thống IoT. Cuối cùng, hệ thống back-end sẽ triển khai các ứng dụng server phục vụ cho việc nhận và xử lý dữ liệu phần cứng trên Internet. Các ứng dụng này bao gồm: database, API server, và web server.

Mô hình hệ thống IoT sử dụng mô hình IoT 4 lớp. Trong đó, sensing layer sử dụng MCU STM32 giao tiếp vạn vật. Trên network layer, hệ thống back-end triển được khai bằng kỹ thuật Docker Containerizing, sử dụng MongoDB database, EggJS API server, và MQTT Broker, để giao tiếp với ESP32 IoT gateway. Trên data processing layer, kỹ thuật giao tiếp được triển khai dựa trên giao thức frame trong giao tiếp gateway-API server, kỹ thuật đồng bộ hoạt động trên sự kiện "change streams" của MongoDB, và kỹ thuật mật mã hóa nhẹ ChaCha20-Poly1305 được triển khai trên gói data của giao thức frame. Trên application layer, Nginx web server được sử dụng để cung cấp giao diện web linh hoạt cho người dùng.

Trên phần cứng, cơ chế giao tiếp và đồng bộ đã triển khai thành công trên MCU STM32 và ESP32. Trên VPS, hệ thống back-end đã được triển khai thành công bằng kỹ thuật Docker Containerizing. Trong đó, cơ chế giao tiếp và đồng bộ được cài đặt trên API server. Về giao diện, giao diện web đã được host thành công trên Nginx web server, có cấu hình PWA, và giao diện có thể tùy chỉnh. Cuối cùng, mật mã hóa nhẹ ChaCha20-Poly1305 đã được triển khai thanh công trên giao tiếp gateway-API server.

Cơ chế giao tiếp giúp tương tác giữa device-gateway-API server trở nên đáng tin cậy và có thể tự phục hồi khi phát hiện lỗi. Về việc đồng bộ, cơ chế động bộ giúp thiết bị phần cứng và giao diện web luôn cập nhật được những thay đổi của các VS trên database. Về mật mã hóa, kỹ thuật mật mã hóa nhẹ giúp giao thức giao tiếp có tính bảo mật trong tương tác gateway-API server. Cuối cùng, giao diện web linh hoạt giúp người dùng dễ dàng thiết kế và tùy chỉnh các widget trên UI dashboard.

Trong giao tiếp giữa phần cứng và API server, giao thức giao tiếp chưa đưa ra nhiều API tương tác với các VS. Hơn nữa, giao thức giao tiếp chưa đưa ra các API giúp người dùng phát sinh các sự kiện trên phần cứng. Tiếp theo, trong tương tác giữa back-end và front-end, giao diện web và API server chưa đưa ra tương tác giúp các quản trị viên thao tác với dữ liệu người dùng. Cuối cùng, giao diện web chưa đưa ra nhiều widget giúp người dùng có thêm lựa chọn trong việc xây dựng các UI dashboard.

Về định hướng phát triển, nền tảng IoT có thể phát triển các tương tác device-server, tương tác với dữ liệu người dùng, và widget xây dựng giao diện web linh hoạt. Đầu tiên, các tương tác device-server có thể phát triển để cung cấp thêm các giao tiếp đọc/ghi những kiểu dữ liệu khác của các VS (string, enumerate, array,...). Tiếp theo, hệ thống có thể cung cấp thêm các API và tương tác web để các quản trị viên quản lý dữ liệu người dùng. Cuối cùng, các widget có thể được tạo mới nhiều hơn để người dùng có thêm nhiều lựa chọn trong việc xây dựng UI dashboard.

DANH MỤC CÔNG TRÌNH CỦA TÁC GIẢ

1. Nguyễn Tiến Đạt,Nguyễn Vũ Minh Thành,Đỗ Đức Phú,Nguyễn Văn Nhị,Lê Đức Hùng, (2022), "Thực hiện thuật toán ChaCha20 - Poly1305 trên phần cứng ứng dụng bảo mật hệ thống IoT", *Hội nghị Quốc gia về Điện tử, Truyền thông và Công nghệ Thông tin lần thứ XXV, REV-ECIT 2022.*

TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] Blynk. *Introduction - Blynk Documentation.* URL: https://docs.blynk.io/en/ (visited on 08/01/2023).

[2] ThingsBoard. *What is ThingsBoard?* URL: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/ (visited on 08/02/2023).

[3] ThingsBoard. *Welcome to the ThingsBoard Devices Library!* URL: https://thingsboard.io/docs/devices-library/ (visited on 08/02/2023).

[4] InterviewBit. *IoT Architecture – Detailed Explanation.* URL: https://www.interviewbit.com/blog/iot-architecture (visited on 07/22/2023).

[5] Nir, Yoav and Langley, Adam. *ChaCha20 and Poly1305 for IETF Protocols.* RFC 7539. May 2015. DOI: 10.17487/RFC7539. URL: https://www.rfc-editor.org/info/rfc7539.

[6] HiveMQ. *The Free Public MQTT Broker.* URL: https://www.hivemq.com/public-mqtt-broker/ (visited on 08/08/2023).

[7] Wikipedia. *Create, read, update and delete.* URL: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete#RESTful_APIs (visited on 08/09/2023).

[8] Mongoose. *Elegant mongodb object modeling for node.js.* URL: https://mongoosejs.com/ (visited on 08/09/2023).

[9] Docker. *What is MongoDB?* URL: https://hub.docker.com/_/mongo (visited on 08/09/2023).

[10] MongoDB. *Replication.* URL: https://www.mongodb.com/docs/manual/replication/ (visited on 08/09/2023).