

ICDR-IoT: Integração e Controle de Dispositivos de Rede via IoT

Marco A. Auth, Pedro H. Salvador

Orientação: Rafael T. Pereira

Redes de Computadores – Universidade Federal de Santa Maria (UFSM)

rafatp@redes.ufsm.br, marco.auth@redes.ufsm.br,
pedro.salvador@redes.ufsm.br

Abstract. This paper presents ICDR-IoT, a system for remote access and control of network devices using Internet of Things (IoT) technologies. An ESP32 microcontroller bridges network administrators and managed switches over a serial (UART/TTL) interface, converted to RS-232 by a MAX3232 module. A Socket.IO web interface communicates with a Python/Flask backend that forwards commands to the ESP32 over TCP and returns switch responses in real time. The architecture was implemented and functionally tested with the switches of Laboratory 306 at CTISM/UFSM; formal quantitative evaluation is left for future work.

Resumo. Este artigo apresenta o ICDR-IoT, um sistema de acesso e controle remoto de dispositivos de rede baseado em Internet das Coisas (IoT). Um microcontrolador ESP32 estabelece a ponte entre administradores de rede e switches gerenciáveis via interface serial (UART/TTL), convertida para RS-232 por um módulo MAX3232. Uma interface web integrada via Socket.IO comunica-se com um backend em Python/Flask, que encaminha comandos ao ESP32 via TCP e retorna as respostas em tempo real. A arquitetura foi implementada e testada funcionalmente com os switches do Laboratório 306 do CTISM/UFSM, com avaliação quantitativa formal deixada para trabalhos futuros.

1. Introdução

A complexidade de configurar diversas estruturas de redes de computadores vem crescendo nos últimos anos; assim, a agilidade e a otimização dessas configurações são cruciais para que exista maior eficiência na utilização desses recursos. Nesse contexto, o projeto ICDR-IoT (Integração e Controle de Dispositivos de Rede via IoT) propõe o desenvolvimento de um sistema baseado em Internet das Coisas (IoT, do inglês Internet of Things) para o acesso e controle remoto dos equipamentos de rede do Laboratório 306 do Colégio Técnico Industrial de Santa Maria (CTISM), da Universidade Federal de Santa Maria (UFSM), com o objetivo de reduzir a necessidade de intervenções presenciais e otimizar a manutenção da infraestrutura. Parte desses equipamentos são switches legados, sem suporte a interfaces modernas de gerência programática com controle de acesso, de modo que a porta de console serial é a interface de configuração comum a todos eles.

O sistema consiste em um dispositivo embarcado baseado no microcontrolador ESP32, que faz a ponte entre administradores e switches por meio de uma UART (Universal Asynchronous Receiver-Transmitter) em nível lógico TTL (Transistor Transistor Logic), e em uma aplicação web em HTML (HyperText Markup Language), CSS (Cascading Style Sheets) e JavaScript. O projeto busca aumentar a eficiência operacional da infraestrutura de TI (Tecnologia da Informação) e servir como

ambiente de capacitação, alinhado ao Plano de Desenvolvimento Institucional (PDI) da UFSM.

2. Trabalhos Relacionados

Soluções comerciais de console server, como a linha CM8100 da Opengear [Opengear 2026] e o SLC 8000 da Lantronix [Lantronix 2026], oferecem acesso serial multiportas com autenticação e criptografia. Não há incompatibilidade técnica entre essas soluções e os equipamentos Cisco, cuja porta de console segue o padrão RS-232 (Recommended Standard 232); a limitação, no contexto deste trabalho, é de custo e de propósito: são equipamentos genéricos, voltados a data centers, com custo por porta elevado (Seção 4) e sem uma interface orientada ao Cisco IOS (Internetwork Operating System) ou ao uso didático. Bridges de baixo custo baseados no ESP8266, como o esp-link [JeeLabs 2026], e ferramentas de código aberto, como o ser2net [Minyard 2026], expõem uma porta serial via rede de forma semelhante ao ICDR-IoT, mas sem interface dedicada ao Cisco IOS. Já os protocolos NETCONF (Network Configuration Protocol) [Enns et al. 2011] e RESTCONF [Bierman et al. 2017] padronizam a configuração por meio de uma API (Application Programming Interface), mas exigem suporte do equipamento, indisponível nos switches legados do Laboratório 306.

O ICDR-IoT se diferencia por combinar hardware de baixo custo com um terminal web dedicado ao Cisco IOS e voltado a laboratórios acadêmicos.

3. Visão Geral do Projeto

O ICDR-IoT adota uma arquitetura de três camadas (Figura 1): o microcontrolador ESP32, um servidor intermediário em Flask [Pallets Projects 2024] com suporte a WebSockets e uma interface web, integrados em tempo real pela biblioteca Socket.IO [Socket.IO 2024].

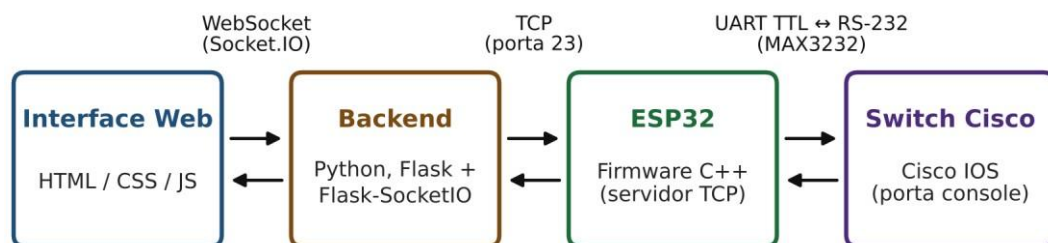


Figura 1. Diagrama de arquitetura em camadas do ICDR-IoT

O administrador seleciona na interface web o ESP32 desejado, pelo seu endereço IP (Internet Protocol); o servidor encaminha cada comando por uma conexão TCP (Transmission Control Protocol) ao ESP32, que o transmite ao switch pela UART e devolve a resposta, repassada ao terminal do navegador, dispensando o acesso físico aos equipamentos.

4. Hardware

O hardware escolhido como elo entre o servidor e o dispositivo de rede foi o ESP32 (ESPWROOM-32), um SoC (System-on-Chip) da Espressif Systems com microprocessador dual-core de 32 bits, Wi-Fi e Bluetooth, clock de até 240 MHz, 520 KiB de RAM (Random Access Memory) e 4 MiB de memória flash [Espressif Systems 2023]. Foi escolhido por sua capacidade de processamento, baixo consumo de energia e versatilidade de interfaces, especialmente a UART.

A comunicação entre o ESP32 e os switches é feita pela UART, em nível lógico TTL, na qual um dispositivo transmite e o outro recebe dados bit a bit. Como os equipamentos de rede operam em nível RS-232, incompatível com a lógica TTL do

ESP32, foi empregado um conversor baseado no circuito integrado MAX3232, que assegura a conversão bidirecional dos sinais [Maxim Integrated 2014].

O hardware evoluiu de componentes em protoboard para uma PCB (Placa de Circuito Impresso) dedicada, que integra o ESP32 e o MAX3232 em um único circuito e elimina fios soltos; sua fabricação já foi concluída, restando os testes elétricos e funcionais de validação. Um invólucro de proteção também já foi impresso por FDM (Fused Deposition Modeling, modelagem por deposição fundida) a partir de um modelo 3D dimensionado pela PCB; esse primeiro protótipo ainda está em validação, e o formato definitivo (espessura de parede e ventilação) segue em ajuste.

A arquitetura atual estabelece uma relação um-para-um entre ESP32 e switch gerenciável: escalar a solução para todos os equipamentos do Laboratório 306 exigiria um módulo dedicado por switch, ainda sem um mecanismo de descoberta automática dos ESP32 ativos na rede (por exemplo, via mDNS, multicast Domain Name System), o que hoje exige o cadastro manual de cada IP na interface web. Em termos de custo, os componentes de cada unidade (ESP32, conversor MAX3232, PCB e invólucro impresso) somam poucas dezenas de dólares, uma estimativa de mercado bem inferior aos US\$ 90–180 por porta de um console server comercial equivalente, como o Opendgear CM8116/CM8132 [Opendgear 2026].

5. Firmware do ESP32 (Arduino C++)

O firmware do ESP32 foi desenvolvido em C++ na IDE (Integrated Development Environment) Arduino e tem como funções a conexão à rede Wi-Fi, a criação de um servidor TCP e a comunicação serial com os dispositivos de rede. Na inicialização, o microcontrolador conecta-se a uma rede sem fio previamente configurada e obtém um endereço IP dinâmico, pelo qual é acessado pelo backend.

Uma vez conectado, o ESP32 inicia um servidor TCP na porta 23 (Telnet), que aguarda conexões do backend Python. Cada instrução recebida é retransmitida à UART2 (pinos GPIO, General-Purpose Input/Output, 16 e 17, a 9600 bps), ligada ao switch pelo conversor TTL/RS-232. Após o envio do comando, o ESP32 permanece em escuta pela UART, coletando a resposta com timeout para evitar bloqueios, e a retransmite ao cliente TCP, atuando como ponte transparente entre a rede Wi-Fi e o dispositivo físico.

6. Backend com Python (Flask + Socket.IO)

O backend foi desenvolvido em Python com Flask [Pallets Projects 2024] e com a extensão Flask-SocketIO [Grinberg 2024], e intermedeia a comunicação entre a interface web e os ESP32. O Flask estrutura a aplicação e fornece as rotas HTTP (HyperText Transfer Protocol); a rota principal carrega a interface para selecionar o ESP32 e enviar comandos. A comunicação em tempo real é feita por WebSockets, via Socket.IO [Socket.IO 2024], sem recarregar a página.

Optou-se pelo Socket.IO em vez de requisições HTTP periódicas (polling) ou de WebSockets nativos porque a biblioteca oferece comunicação bidirecional orientada a

eventos, reconexão automática do cliente, transporte alternativo por long-polling e integração direta com o Flask; já o MQTT (Message Queuing Telemetry Transport), comum em IoT, exigiria um broker adicional e uma ponte até o navegador. Para o enlace com o microcontrolador, o servidor abre uma conexão TCP na porta 23 do ESP32, como um cliente Telnet, e usa threads dedicadas para que essas operações bloqueantes não travem o atendimento a outros clientes.

7. Frontend (HTML + JavaScript)

A interface web é um painel único (dashboard), com blocos laterais e terminal central

(Figura 2). O bloco “Conexão” contém o IP do ESP32, o histórico de IPs e os botões Conectar/Encerrar; o bloco “Sessão” exibe status, host, porta, protocolo, comandos enviados e tempo de sessão (uptime).

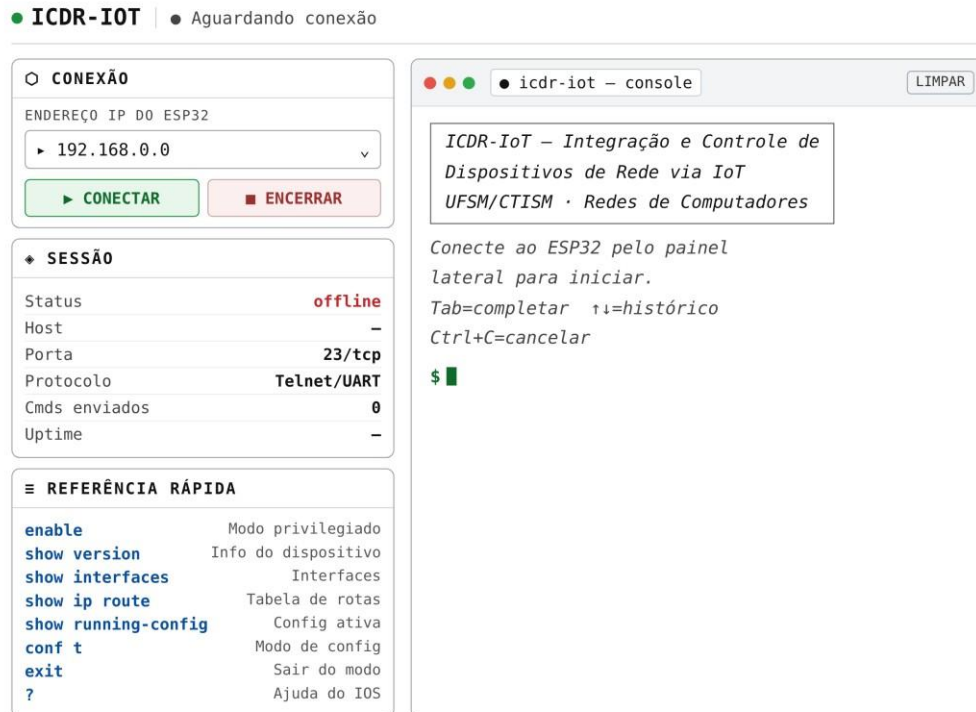


Figura 2. Interface web do ICDR-IoT: painéis de conexão, sessão, referência rápida e terminal

Um terceiro bloco, “Referência Rápida”, lista os comandos mais usados do Cisco IOS [Cisco Systems 2020], como enable, show version, show running-config e configure terminal, cada um com uma breve descrição, como guia para usuários menos familiarizados com o IOS. A área central traz um terminal estilizado, com cabeçalho da sessão ativa e botão de limpeza; comandos digitados são enviados pelo evento send_command e as respostas chegam pelo evento command_response.

O terminal oferece autocompletar de comandos IOS pela tecla Tab, histórico de comandos navegável pelas setas e histórico de IPs no bloco de conexão; o atalho Ctrl+C cancela a digitação, aproximando a experiência de um terminal Cisco tradicional.

8. Integração Entre as Partes

A interação ocorre sobre três enlaces: WebSocket (navegador↔backend), TCP (backend↔ESP32, porta 23) e UART (ESP32↔switch, via TTL/RS-232), conforme o diagrama de sequência da Figura 3. Ao clicar em Conectar, o frontend emite o evento connect_to_esp; o Flask-SocketIO cria uma thread dedicada que abre um socket TCP para o ESP32 e confirma a conexão ao navegador, mantendo escuta contínua sem bloquear o servidor. Ao clicar em Enviar, o frontend emite send_command; o backend escreve o comando no socket TCP, terminado por \n, e o ESP32 o retransmite ao switch pela UART2, coletando a saída caractere a caractere com timeout dinâmico.

A thread do Flask decodifica os bytes recebidos e emite o evento command_response, e o frontend acrescenta o texto ao terminal com rolagem automática, simulando uma sessão Telnet. Ao clicar em Desconectar, ou diante de erro, o frontend emite disconnect_esp; o backend fecha o socket e o ESP32 volta a aguardar conexões.

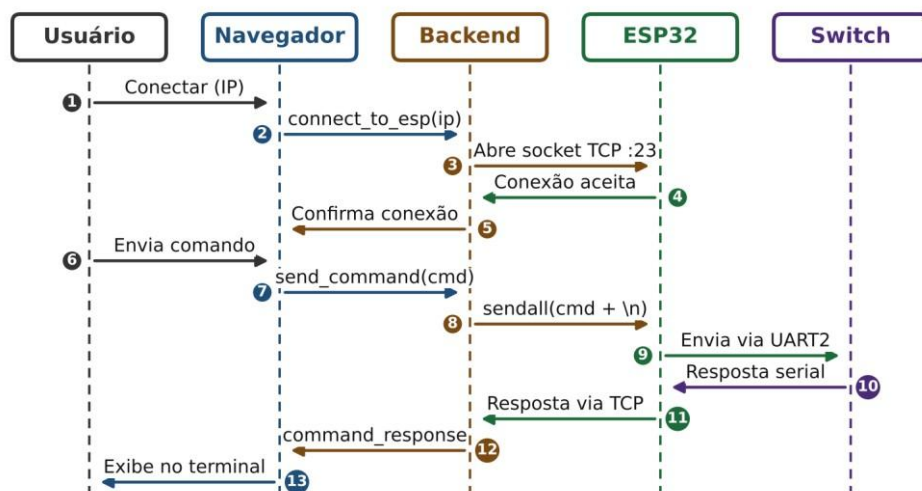


Figura 3. Diagrama de sequência da comunicação entre usuário, navegador, backend, ESP32 e switch

9. Testes e Resultados

Além dos testes funcionais pela interface web (Seção 8), um script Python (`switch_benchmark.py`) mediu a latência do sistema conectando-se diretamente ao ESP32 via TCP/Telnet (porta 23), enviando comandos Cisco IOS pré-definidos e registrando o tempo decorrido até a resposta em arquivo CSV (Comma-Separated Values). Nas primeiras rodadas (perfis básico e diagnóstico), nenhuma resposta foi capturada: as leituras retornaram vazias após o silêncio de 400 ms, valor subdimensionado para o switch. Após o ajuste do timeout, a rodada seguinte (resistência) capturou respostas reais, entre 1104 ms e 7296 ms.

A Tabela 1 resume as 11 medições de latência por comando obtidas nessa rodada. O tempo médio de resposta foi de 3580,18 ms (mediana de 3110,04 ms), com alta dispersão (desvio padrão de 2450,26 ms) entre comandos simples, como `show clock`, e comandos de saída longa, como `show running-config` e `show version`.

Tabela 1. Estatísticas de latência por comando (n = 11 medições)

Métrica	Valor	Métrica	Valor
Mínimo	1104,46 ms	Percentil 95 (P95)	7170,44 ms
Média	3580,18 ms	Percentil 99 (P99)	7270,70 ms
Mediana	3110,04 ms	Máximo	7295,76 ms
Desvio padrão	2450,26 ms	Bytes médios/resposta	310 B
		Vazão média	≈100 B/s

Nas saídas mais longas (`show version`, `show ip interface brief`, `show interfaces status`), observou-se perda e corrupção de caracteres no texto capturado, indicando perda de bytes na retransmissão pela UART sob rajadas de dados. O comando `show runningconfig` retornou o erro do Cisco IOS `"% Invalid input detected at '^' marker"`, sugerindo corrupção no início da string enviada. Os resultados confirmam a viabilidade funcional do sistema, mas revelam uma limitação de confiabilidade na UART sob maior volume de dados. Uma validação mais ampla, com repetições por comando em operação contínua, está planejada.

Do ponto de vista de segurança, há limitações relevantes para uso além do ambiente de testes: o servidor TCP do ESP32 (porta 23) aceita comandos sem

autenticação, de modo que qualquer dispositivo na rede do laboratório pode enviar comandos arbitrários, inclusive em modo privilegiado (enable), diretamente ao switch; o backend não valida a sessão do usuário; nenhum comando é registrado com usuário ou horário; e conexões TCP órfãs, quando o navegador é fechado abruptamente, podem manter a porta 23 ocupada até um reinício manual. Também não há isolamento de rede, como uma VLAN (Virtual Local Area Network) dedicada, nem caracterização do comportamento diante de duas conexões simultâneas ao mesmo ESP32.

10. Conclusão

Este artigo apresentou o ICDR-IoT, um sistema de baixo custo para acesso remoto aos switches do Laboratório 306 do CTISM/UFSM, baseado em ESP32, backend Flask/Socket.IO e interface web.

Os testes confirmaram o fluxo de comunicação entre as camadas (3,58 s de latência média), mas revelaram perda de confiabilidade na UART e lacunas de segurança (Seção 9). A PCB e o invólucro 3D já foram fabricados, restando sua validação final. Prioriza-se como trabalho futuro (i) a implementação de um mecanismo de autenticação e segurança na porta 23 e na interface web, já que comandos privilegiados hoje chegam a switches reais sem proteção; seguem-se (ii) o bloqueio de comandos destrutivos; (iii) o registro (log) de comandos; (iv) o ajuste do tratamento de buffer da UART do ESP32 para eliminar a perda de caracteres; e (v) a reconexão automática de Wi-Fi.

Referências

- Analog Devices. “MAX3232: 3.0V to 5.5V, Low-Power, Up to 1Mbps, True RS-232 Transceivers”. <https://www.analog.com/en/products/max3232.html>.
- Bierman, A., Bjorklund, M. and Watsen, K. (2017) “RESTCONF Protocol”. RFC 8040, Internet Engineering Task Force (IETF). <https://www.rfc-editor.org/rfc/rfc8040>.
- Cisco Systems. “Cisco IOS Configuration Guides”. <https://www.cisco.com/c/en/us/support/ios-nx-os-software/ios-software/products-installation-and-configuration-guides-list.html>.
- Enns, R., Bjorklund, M., Schoenwaelder, J. and Bierman, A. (2011) “Network Configuration Protocol (NETCONF)”. RFC 6241, Internet Engineering Task Force (IETF). <https://www.rfc-editor.org/rfc/rfc6241>.
- Espressif Systems. “ESP32-WROOM-32 Datasheet”. https://documentation.espressif.com/esp32-wroom-32_datasheet_en.html.
- Grinberg, M. “Flask-SocketIO Documentation”. <https://flask-socketio.readthedocs.io>.
- JeeLabs. “esp-link: ESP8266 Wi-Fi-Serial Bridge”. <https://github.com/jeelabs/esp-link>.
- Lantronix. “SLC 8000 Advanced Console Manager”. <https://www.lantronix.com/products/lantronix-slc-8000/>.
- Minyard, C. “ser2net: Serial to Network Interface”. <https://github.com/cminyard/ser2net>.
- Opengear. “CM8100 Console Server”. <https://opengear.com/products/cm8100-console-server/>.
- Socket.IO. “Socket.IO Documentation, version 4”. <https://socket.io/docs/v4/>.