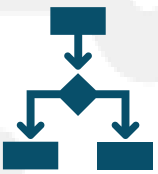
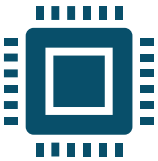
Roberto Higino Pereira da Silva
Joésia Moreira Julião Pacheco
Joelma Monteiro de Carvalho
Organizadores

Curso de Capacitação Profissional Técnica

Hardware e Firmware IoT com IMPC HTNB32L

Módulo 3: Conectando o Futuro com NB-IoT


Manaus-AM

2025

EQUIPE TÉCNICA

Diretor:

Raimundo Cláudio Souza Gomes

Coordenador Geral

Roberto Higino Pereira da Silva

Coordenadores Pedagógicos:

Joésia Moreira Julião Pacheco

Joelma Monteiro de Carvalho

Professores

Celso Barbosa Carvalho

Rafael Facioni Scalabrin

Apoio Técnio Hana Electronics

Weslley Fábio Ferreira Santos

SUMÁRIO

Apresentação

Oferecido pela Universidade do Estado do Amazonas (UEA), por meio da Escola Superior de Tecnologia (EST) e do Hub – Tecnologia e Inovação, o curso de extensão "Configuração e Operação da Rede NB-IoT do iMCP HTNB32L", com carga horária de 30 horas, teve como principal objetivo capacitar os estudantes na configuração, operação e diagnóstico de redes NB-IoT, utilizando o módulo iMCP HTNB32L como plataforma base.

O curso abordou de forma integrada a teoria e a prática sobre a arquitetura, os protocolos e os modos de conectividade característicos das redes Narrowband IoT (NB-IoT), permitindo que os participantes compreendessem não apenas os fundamentos técnicos, mas também a aplicação desses conhecimentos no desenvolvimento de soluções embarcadas. As atividades práticas incluíram o envio de dados de sensores para plataformas IoT por meio de protocolos leves, como MQTT e CoAP, com foco em práticas de versionamento de código e documentação técnica — aspectos fundamentais para o desenvolvimento de projetos robustos e sustentáveis no contexto da Internet das Coisas.

Ao proporcionar esse aprendizado aplicado, o curso também promoveu o protagonismo estudantil, alinhando-se à missão da extensão universitária de articular ensino e prática com impacto social e tecnológico. Os estudantes puderam vivenciar uma formação voltada à inovação, à autonomia técnica e ao desenvolvimento de competências essenciais para o mercado atual, cada vez mais demandante por soluções inteligentes e conectadas.

Além disso, ao ser conduzido em um ambiente multidisciplinar, o curso incentivou a colaboração entre áreas do conhecimento e fortaleceu a integração entre a universidade e os desafios tecnológicos contemporâneos. Assim, contribuiu significativamente para o amadurecimento acadêmico e profissional dos participantes, demonstrando o papel fundamental da universidade como promotora de conhecimento aplicado, inovação e transformação social.

Coordenador Geral

Professor Roberto Higino Pereira da Silva

Módulo 3: Conectando o Futuro com NB-IoT

Neste módulo, será explorado as engrenagens que tornam possível a comunicação eficiente e de longo alcance entre dispositivos no mundo da Internet das Coisas.

O iMCP HTNB32L, equipado com tecnologia NB-IoT, é uma poderosa ferramenta para aplicações em cidades inteligentes, monitoramento remoto, rastreamento de ativos e muito mais. Aqui, será discutido como configurar, operar e aproveitar ao máximo esse recurso para transformar ideias em soluções reais e conectadas.

Objetivo:

Capacitar o aluno a configurar e operar a conectividade NB-IoT no iMCP HTNB32L, compreendendo os fundamentos da rede, os comandos AT utilizados, e os procedimentos necessários para estabelecer comunicação com servidores remotos em aplicações de Internet das Coisas

Carga horária: 30 horas

Capítulos deste Módulo:

Capítulo 1: Conexão com Rede NBIoT comando AT

Capítulo 2: Comunicação via SOKET

Capítulo 3: Conexão NB via Firmware

Capítulo 4: Conexão MQTT via Firmware

Capítulo 5: Conexão HTTP via Firmware

Capítulo 6: Configuração EDRX e PSM via Comando AT

Capítulo 7: EDRX e PSM via Firmware

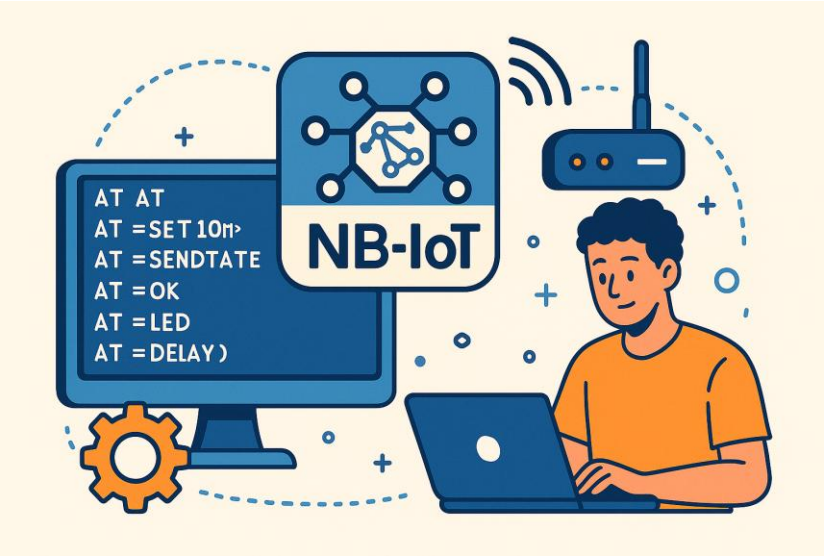
Capítulo 1: Conexão com Rede NBIoT comando AT

Antes da implementação de bibliotecas e códigos complexos, o domínio dos comandos AT é fundamental para a comunicação NB-IoT. Estes comandos, por sua natureza direta, possibilitam a configuração do modem, o registro na rede da operadora, e o envio e recebimento de dados, tudo isso por meio de uma interface de terminal.

Neste capítulo, será explorada a interação com o módulo NB-IoT do HTNB32L utilizando exclusivamente comandos AT, abrangendo os seguintes tópicos:

- Verificação do estado da conexão.
- Obtenção de informações do dispositivo (IMEI, DevEUI).
- Registro na rede da operadora.
- Configuração de parâmetros como bandas e operadores.
- O domínio dos comandos AT é essencial para o controle do dispositivo.

Figura 1: Sua primeira conexão com o mundo NB-IoT



Fonte: Lab 4.1 (Equipe técnica do projeto)

1.1. Introdução à Conexão NB-IoT via Comandos AT

A comunicação NB-IoT pode ser estabelecida utilizando comandos AT, uma forma direta e eficaz de configurar e interagir com o modem do iMCP HTNB32L. Esses comandos permitem inicializar o módulo, verificar o status da rede, configurar parâmetros de conexão e iniciar a comunicação com a operadora.

Os comandos AT mais utilizados nesta etapa são:

- AT+CFUN=1: Liga o modem.
- AT+COPS?: Verifica a operadora conectada.
- AT+CSQ: Verifica a qualidade do sinal.
- AT+CGATT=1: Anexa o dispositivo à rede NB-IoT.

Esses comandos são enviados por terminal serial e representam o primeiro passo para validar a conectividade com a rede NB-IoT antes da implementação por firmware.

Verificação da Qualidade de Sinal e Registro na Rede

Após ativar o modem, é essencial verificar se o dispositivo está registrado na rede NB-IoT e se há qualidade de sinal suficiente para a comunicação. Isso é feito com os seguintes comandos AT:

- **AT+CSQ** – Retorna a qualidade do sinal em dBm. Valores recomendados: entre 10 e 30.
- **AT+CGATT?** – Verifica se o dispositivo está anexado à rede. Retorno 1 indica sucesso.
- **AT+CEREG?** – Verifica o status de registro na rede. O retorno 0,5 ou 0,1 indica registro bem-sucedido.
- **AT+CGDCONT?** – Confirma a configuração do APN (Access Point Name) utilizado.

1.2. Configurar Bandas e Operadoras

O NB-IoT pode operar em diferentes bandas de frequência, e é importante configurá-las corretamente para garantir a conexão com a rede. Os comandos usados incluem:

- AT+NBAND=? → Lista as bandas disponíveis.
- AT+NBAND=<banda> → Seleciona a banda desejada.
- AT+CGDCONT=1,"IP","<APN>" → Define o ponto de acesso (APN) da operadora.

Exemplo de configuração:

```
AT+NBAND=8
AT+CGDCONT=1,"IP","nb.inetd.gd"
```

Esses parâmetros devem ser ajustados conforme a operadora local que oferece suporte à tecnologia NB-IoT.

Figura 2: Configurando Bandas e Operadoras

Fonte: Lab 4.1 (Equipe técnica do projeto)

Capítulo 2: Conectando Dispositivos ao Mundo com Sockets

Na era da Internet das Coisas, a conectividade de dispositivos à rede constitui um requisito técnico fundamental para a inovação. Neste capítulo, será explorado o conceito de comunicação via socket, uma ferramenta essencial para a transmissão direta, segura e eficiente de dados entre dispositivos.

Os sockets permitem a comunicação entre dispositivos e servidores, possibilitando o envio de medições de sensores, o controle de atuadores remotos e a construção de aplicações conectadas.

Prepara-se para o domínio de um dos pilares da comunicação em redes NB-IoT e para o avanço no desenvolvimento de soluções IoT robustas e inteligentes.

Figura 3: Conectando Dispositivos ao Mundo com Sockets

Fonte: Lab 4.1 (Equipe técnica do projeto)

2.1. Conceito de Comunicação via SOCKET no contexto NB-IoT

A comunicação via SOCKET no NB-IoT funciona como uma ponte entre o dispositivo embarcado e a rede de dados da operadora, permitindo o envio e recebimento de mensagens diretamente por meio da camada IP.

Principais pontos:

- SOCKET é uma interface de software que permite que dois dispositivos se comuniquem por meio de protocolos como UDP ou TCP.
- No contexto do HTNB32L, a comunicação via SOCKET é realizada usando comandos AT específicos fornecidos pelo modem LTE.
- Essa abordagem é útil para aplicações que não utilizam protocolos de alto nível como MQTT ou HTTP, mas ainda precisam de uma comunicação leve e eficiente com servidores remotos.
- A abertura, envio e fechamento de conexões SOCKET são controladas por comandos AT como AT+NSOCR, AT+NSOST, AT+NSORF, e AT+NSOCL.

Esse tipo de comunicação é bastante utilizado em soluções onde o tráfego de dados precisa ser direto, simples e rápido, como em telemetria, sensores ambientais, ou sistemas de alerta.

Comandos AT para uso de SOCKET no NB-IoT

O uso de SOCKETs no HTNB32L por comandos AT segue uma sequência lógica para estabelecer e gerenciar conexões UDP (mais comum no NB-IoT).

Principais comandos:

Comando	Descrição
AT+NSOCR="DGRAM",17,<porta>	Cria um socket UDP e retorna o identificador (ID)
AT+NSOST=<id>,<IP>,<porta>,<tamanho>,<dados>	Envia dados para um endereço IP e porta específicos
AT+NSORF=<id>,<tamanho>	Lê dados recebidos no socket
AT+NSOCL=<id>	Fecha o socket aberto

Exemplo prático:

1. **Criar o socket:**

```
AT+NSOCR="DGRAM",17,3005
```

2. **Enviar mensagem "Olá":**

```
AT+NSOST=0,"187.34.23.11",3000,6,4F6CC3A120
```

3. Fechar o socket:

```
AT+NSOCL=0
```

Esses comandos permitem comunicação ponto-a-ponto com servidores UDP, útil para testes, integrações simples e baixo consumo de dados.

Testando a Comunicação via SOCKET com Servidor Remoto

Com os comandos AT já conhecidos, é possível testar uma comunicação real entre o dispositivo NB-IoT e um servidor na internet.

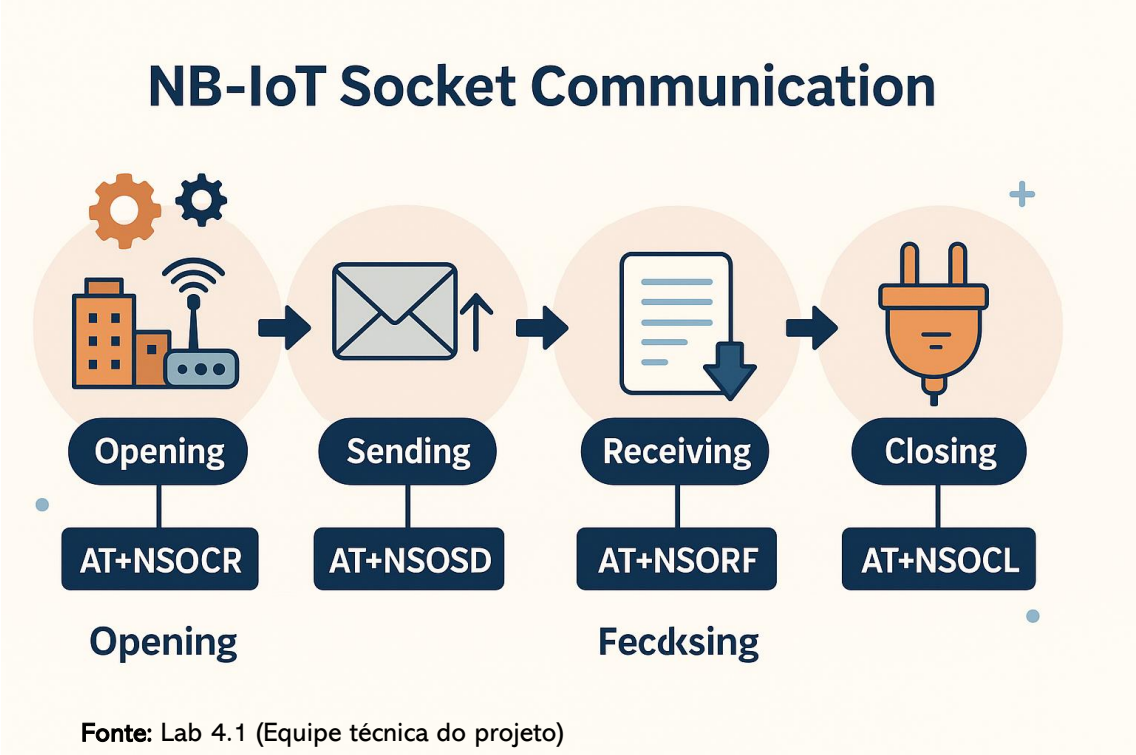
Etapas do teste:

1. Crie o socket UDP com AT+NSOCR, definindo a porta local (por exemplo, 3005).
2. Escolha um servidor UDP público ou local para receber os dados (ex: servidor echo, ou uma aplicação personalizada com netcat, Node.js ou Python).
3. Use AT+NSOST para enviar uma mensagem com dados em hexadecimal. É possível converter strings para hex online ou com scripts.
4. No servidor, verifique se a mensagem foi recebida.
5. Envie uma resposta ao dispositivo e use AT+NSORF para ler os dados recebidos.
6. Finalize a conexão com AT+NSOCL.

Dica prática: Use ferramentas como o Hercules Setup Terminal, netcat (nc) ou servidores em Python para simular um endpoint UDP e visualizar os dados.

Esse teste comprova que o dispositivo está conectado à internet via NB-IoT e é capaz de trocar dados com servidores remotos, sem necessidade de protocolos de aplicação.

Figura 4: NBIoT- Socket Communication



Fonte: Lab 4.1 (Equipe técnica do projeto)

2.2. Conversão de Dados para Envio via SOCKET (Hexadecimal)

O comando AT+NSOST, utilizado para envio de dados por socket, exige que os dados sejam enviados no formato hexadecimal, mesmo que originalmente sejam strings ou números.

Exemplo:

Para enviar a mensagem "HELLO" via socket, é necessário converter cada caractere para seu valor hexadecimal ASCII:

Caractere	Código HEX
H	48
E	45
L	4C
L	4C
O	4F

Resultado: 48454C4C4F

Como converter:

- **Online:** Use ferramentas como RapidTables
- Python:

```
"HELLO".encode("utf-8").hex()
```

Envio com AT+NSOST:

```
AT+NSOST=0,"187.34.23.11",3000,5,48454C4C4F
```

Esse passo é essencial para garantir que o servidor remoto receba os dados corretamente.

Finalizando a Comunicação: Fechando o Socket

Após enviar e/ou receber os dados com sucesso via NB-IoT, é fundamental encerrar a conexão socket corretamente para liberar os recursos do modem e evitar conflitos em futuras transmissões.

Comando para fechamento:

```
AT+NSOCL=<id>
```

<id> corresponde ao identificador do socket que foi criado com o comando AT+NSOCR.

Exemplo:

```
AT+NSOCL=0
```

Por que é importante fechar o socket?

- Libera memória e recursos internos do modem.
- Evita falhas de conexão em novos envios.
- Garante o bom funcionamento do ciclo de conexão.

Em sistemas embarcados com recursos limitados, uma boa prática é fechar o socket imediatamente após a comunicação, principalmente quando o envio de dados é esporádico.

Capítulo 3: Conexão NBIoT via firmware

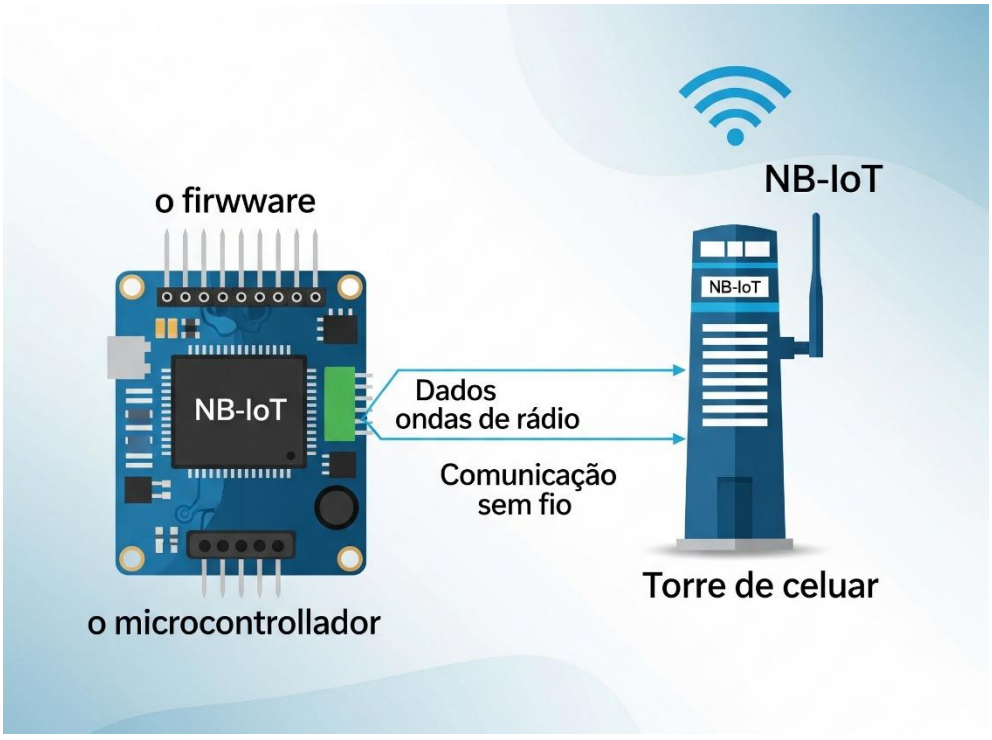
A Internet das Coisas (IoT) propõe a interconexão de objetos e sistemas para o monitoramento de status, otimização operacional e automação inteligente. Nesse contexto, o NB-IoT é uma tecnologia fundamental para aplicações que demandam baixa potência e longo alcance.

O firmware constitui um componente crítico para a efetivação da comunicação em dispositivos IoT. Ele compreende o conjunto de instruções programadas que capacitam o hardware a operar e interagir no ambiente digital.

A programação do firmware para conectividade NB-IoT permite o controle funcional do dispositivo. Essa habilidade técnica é essencial para o desenvolvimento de soluções IoT que requerem a transmissão de dados e a integração em redes de comunicação.

O domínio dessa programação capacita o desenvolvedor a gerenciar a conexão diretamente no código, assegurando a funcionalidade e o desempenho dos dispositivos embarcados.

Figura 5: Conexão NBIoT via Firmware



Fonte: Lab 4.1 (Equipe técnica do projeto)

3.1. Fundamentos da Conexão NB-IoT no Firmware

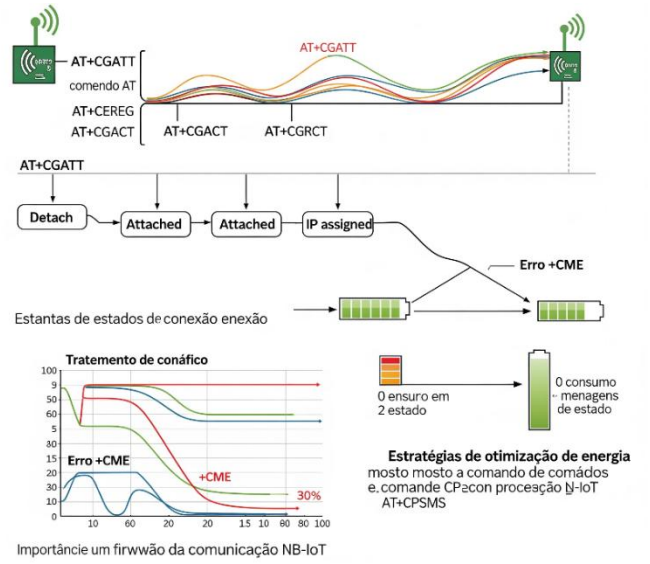
Para que um dispositivo se conecte à rede NB-IoT, o firmware precisa gerenciar uma série de operações essenciais. O hardware responsável por essa comunicação é o módulo NB-IoT. O firmware interage com este módulo principalmente através de comandos AT (AT Commands), que são um conjunto padronizado de instruções para controlar modems celulares.

Por exemplo, comandos como AT+COPS? permitem verificar a operadora de rede, enquanto AT+CEREG? é usado para verificar o registro na rede. Para estabelecer uma conexão de dados, comandos específicos como AT+QMTOPEN (para abrir uma conexão MQTT) e AT+QMTPUB (para publicar dados) são utilizados.

O firmware também deve monitorar os estados da conexão, como Offline, Conectando, Registrado na Rede, Conectado ao Servidor e Erro. O tratamento de erros é crucial; o firmware precisa ser capaz de identificar falhas de conexão e implementar retentativas ou mecanismos de recuperação.

Além disso, para dispositivos alimentados por bateria, a otimização de energia é fundamental. O firmware pode gerenciar modos de baixo consumo como PSM (Power Saving Mode) e eDRX (extended Discontinuous Reception), que permitem que o dispositivo "durma" por longos períodos, acordando apenas para enviar ou receber dados, prolongando significativamente a vida útil da bateria.

Figura 5: Fundamentos da Conexão NB-IoT no Firmware



Fonte: Lab 4.1 (Equipe técnica do projeto)

3.2. Estrutura Básica de um Firmware para NB-IoT

A construção de um firmware robusto para NB-IoT segue uma lógica sequencial, mas com flexibilidade para tratamento de eventos e erros. Um fluxo básico incluiria:

1. **Inicialização:** Configuração do microcontrolador, das portas de comunicação (geralmente UART para interface com o módulo NB-IoT) e dos pinos de controle do módulo.
2. **Verificação do Módulo:** Envio de um comando AT simples (e.g., AT) para garantir que o módulo NB-IoT está respondendo.
3. **Registro na Rede:** Uma sequência de comandos AT para que o módulo se registre na rede da operadora (e.g., configurar APN, tentar registro).
4. **Estabelecimento da Conexão:** Abrir uma sessão de comunicação com um servidor remoto, que pode ser via TCP/UDP ou, mais comumente para IoT, via protocolo MQTT. Isso envolve comandos para configurar o servidor e a porta.
5. **Envio de Dados:** Formatar os dados (de sensores, por exemplo) e enviá-los usando os comandos AT apropriados (e.g., AT+QMTPUB para MQTT).
6. **Recebimento de Dados (Opcional):** Implementar uma lógica para escutar e processar dados ou comandos vindos do servidor.
7. **Gerenciamento de Energia:** Ativar e gerenciar os modos PSM/eDRX conforme a necessidade de envio de dados, colocando o dispositivo em "modo de sono" para economizar energia.
8. **Tratamento de Erros e Retentativas:** Em cada etapa, implementar mecanismos para detectar falhas e tentar novamente a operação.

Essa estrutura forma a base para um firmware que garante a comunicação eficiente e resiliente de um dispositivo NB-IoT.

Desafios e Boas Práticas na Programação do Firmware

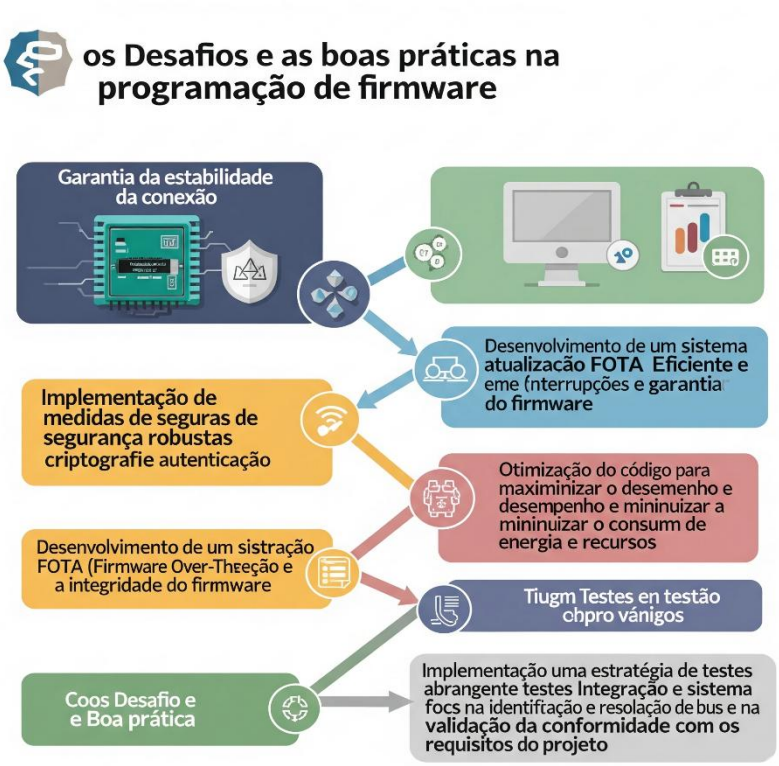
Desenvolver firmware para NB-IoT apresenta desafios únicos que exigem boas práticas para garantir a confiabilidade e longevidade dos dispositivos:

- **Estabilidade da Conexão:** A rede celular pode ser imprevisível. O firmware deve ser projetado para lidar com quedas de sinal, interrupções e reconexões

automáticas para manter a estabilidade da conexão. Implementar watchdogs de hardware e software é uma boa prática.

- **Segurança:** A comunicação deve ser segura. Embora a criptografia e a autenticação possam ser tratadas em camadas superiores (MQTT TLS/SSL, por exemplo), o firmware é a base. Garantir que as chaves e credenciais sejam armazenadas de forma segura e que as comunicações sejam criptografadas quando possível é vital.
- **Atualizações de Firmware (FOTA - Firmware Over-The-Air):** A capacidade de atualizar o firmware remotamente é essencial para corrigir *bugs*, adicionar funcionalidades e aplicar patches de segurança. Um firmware bem planejado inclui um mecanismo para FOTA.
- **Otimização de Código e Memória:** Módulos NB-IoT e microcontroladores para IoT geralmente têm recursos limitados de memória e processamento. O firmware deve ser otimizado para ser leve e eficiente.
- **Testes e Depuração:** Devido à natureza assíncrona da comunicação celular e à latência da rede, testes exaustivos em diferentes cenários de cobertura são cruciais. Ferramentas de depuração e logs detalhados no firmware são indispensáveis.

Figura 6: Desafios e Boas Práticas na Programação do Firmware



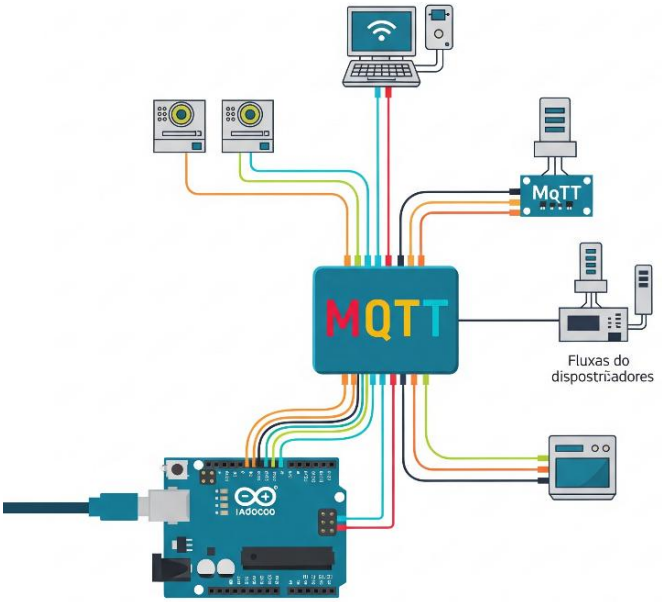
Fonte: Lab 4.1 (Equipe técnica do projeto)

Capítulo 4: Comunicação MQTT via Firmware

Nesse capítulo, será discutida a arquitetura por trás dessa comunicação mágica, será compreendido os papéis de brokers, publishers e subscribers, e como implementar tudo isso diretamente no código que reside no coração do hardware.

Ao final deste capítulo, serão compreendidas as ferramentas para capacitar os projetos com uma conectividade robusta, confiável e pronta para escalar.

Figura 7: Desvendando a Conectividade: O Poder do MQTT no Coração do seu Dispositivo!



Fonte: Lab 4.1 (Equipe técnica do projeto)

4.1. Implementando MQTT no Firmware: Dando Voz ao Seu Dispositivo

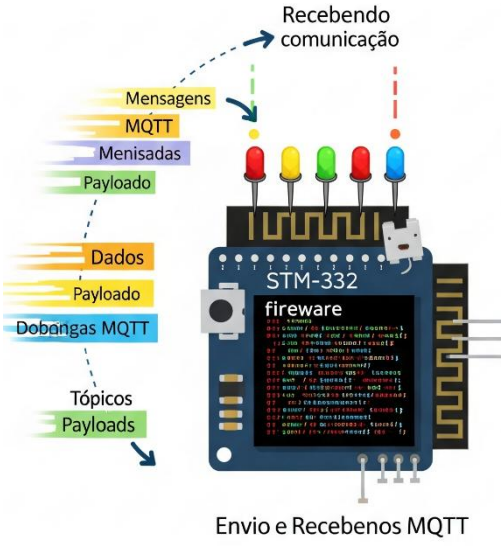
A compreensão teórica do protocolo MQTT precede sua implementação prática em dispositivos embarcados. A integração de MQTT no firmware confere ao hardware a capacidade de se conectar a um broker, enviar dados e receber comandos.

Para simplificar essa integração, a comunidade de desenvolvimento de IoT oferece diversas bibliotecas e Software Development Kits (SDKs). A seleção da biblioteca é geralmente determinada pelo microcontrolador e pela plataforma de desenvolvimento utilizados.

Por exemplo, para os microcontroladores ESP32 e ESP8266, frequentemente empregados em projetos IoT devido à sua conectividade Wi-Fi embarcada, a biblioteca PubSubClient no ambiente Arduino IDE representa uma opção amplamente utilizada. Esta biblioteca é reconhecida por sua leveza, facilidade de uso e cobertura das funcionalidades MQTT essenciais.

Outras alternativas incluem bibliotecas MQTT específicas de fabricantes, como a AWS IoT Device SDK para ESP32 (aplicável em conexões com a nuvem AWS), ou implementações mais genéricas para FreeRTOS, cuja escolha dependerá da complexidade do projeto

Figura 8: Implementando MQTT no Firmware



Fonte: Lab 4.1 (Equipe técnica do projeto)

Configuração da Conexão ao Broker

Antes de qualquer comunicação, seu dispositivo precisa saber para onde enviar e de onde receber as mensagens. Isso envolve configurar os parâmetros de conexão ao seu broker MQTT. Basicamente, será necessário informar ao firmware:

- Endereço do Broker: Pode ser um endereço IP (ex: 192.168.1.100) ou um nome de domínio (ex: broker.hivemq.com).
- Porta do Broker: A porta padrão para MQTT não criptografado é 1883. Para conexões seguras (MQTT over SSL/TLS), a porta padrão é 8883.
- Credenciais de Autenticação (Opcional, mas Recomendado): Muitos brokers exigem um nome de usuário e senha para autenticar a conexão, garantindo que apenas dispositivos autorizados possam se conectar.
- ID do Cliente (Client ID): Um identificador único para seu dispositivo na rede MQTT. Cada cliente conectado ao broker deve ter um Client ID exclusivo.

Exemplo de Código: Publicando Dados (Envio de Leituras de Sensor).

Uma das tarefas mais comuns em IoT é enviar dados de sensores para um sistema central ou para outros dispositivos. Com MQTT, isso é feito publicando mensagens em tópicos específicos. Pense nos tópicos como canais de comunicação temáticos.

Para publicar dados, simplesmente escolhe-se um tópico relevante (ex: sala/temperatura, sensor/umidade/cozinha), formata seus dados (geralmente como JSON ou texto simples) e envia a mensagem.

Exemplo de Código: Assinando Tópicos e Recebendo Comandos

Além de enviar dados, seus dispositivos frequentemente precisam receber comandos ou informações de outros dispositivos. Isso é feito assinando tópicos. Quando uma mensagem é publicada em um tópico que seu dispositivo assinou, o broker a encaminha para ele.

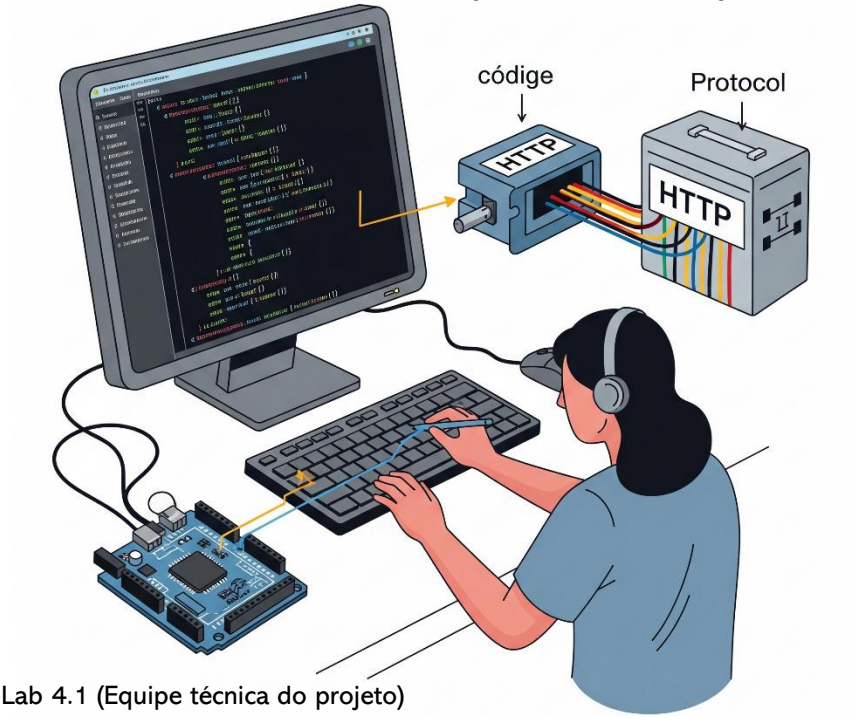
Para lidar com as mensagens recebidas, geralmente define-se uma função de callback que é chamada sempre que uma nova mensagem chega. Dentro dessa função, pode-se analisar o tópico e o conteúdo da mensagem para determinar a ação apropriada (ex: ligar uma luz, ajustar uma configuração).

Capítulo 5: Desbravando a Web: A Comunicação HTTP no Coração do seu Firmware

A comunicação MQTT é reconhecida por sua eficiência na troca de mensagens leves. No entanto, a interação de dispositivos com padrões web estabelecidos e a necessidade de acesso a APIs, envio de dados para servidores complexos ou disponibilização de interfaces de usuário diretamente do firmware demandam outros protocolos. Nesse contexto, o HTTP (Hypertext Transfer Protocol) surge como uma solução para ampliar as possibilidades de projetos embarcados.

Neste capítulo, será explorado como os dispositivos podem operar como "clientes web", capazes de realizar requisições e processar respostas HTTP diretamente do firmware. Serão desmistificados os métodos HTTP, como GET, POST e PUT. Adicionalmente, será abordada a estruturação de requisições e a interpretação de respostas, com o auxílio de exemplos práticos de código adaptáveis a diferentes projetos. O domínio dessas capacidades permitirá que os dispositivos interajam ativamente com o ecossistema da internet, tornando-os participantes da web moderna.

Figura 9: Desbravando a Web: A Comunicação HTTP no Coração do seu Firmware



Fonte: Lab 4.1 (Equipe técnica do projeto)

5.1. Implementando Requisições HTTP no Firmware: Seu Dispositivo como Cliente Web

Agora que são compreendidos os fundamentos do HTTP, é hora de colocar o dispositivo para interagir ativamente com a web, buscando e enviando informações. Isso significa transformar o firmware em um cliente HTTP, capaz de fazer requisições e processar respostas, exatamente como um navegador web faria, mas de forma programática e otimizada para microcontroladores.

Bibliotecas e Abstrações Comuns

A maioria dos microcontroladores com conectividade Wi-Fi ou Ethernet já possui bibliotecas e SDKs que simplificam enormemente a tarefa. Para plataformas populares como ESP32 e ESP8266, a biblioteca HTTPClient (que faz parte do ecossistema Arduino/ESP-IDF) é a ferramenta padrão e altamente eficiente para essa finalidade. Ela abstrai a complexidade das camadas de rede, permitindo que se concentre na lógica da requisição e resposta. Outras plataformas podem ter suas próprias implementações, como as APIs do lwIP ou SDKs específicos de fabricantes.

Fazendo Requisições GET: Acessando Dados de um Servidor

O método GET é o mais comum e serve para solicitar dados de um servidor. Pense em seu dispositivo como um explorador que busca informações em um site ou API. Ao fazer uma requisição GET, deve-se especificar a URL do recurso que deseja e o servidor retorna os dados.

Exemplo Prático (Conceitual para HTTPClient com ESP32/ESP8266):

```
#include <WiFi.h> // Ou <ESP8266WiFi.h> para ESP8266
#include <HTTPClient.h> // Biblioteca para requisições HTTP

const char* ssid = "SEU_SSID";
const char* password = "SUA_SENHA";
const char* apiHost = "api.open-meteo.com";
const int apiPort = 80; // Porta padrão HTTP

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando ao WiFi...");
  }
  Serial.println("WiFi Conectado!");
}

void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http; // Objeto para a requisição HTTP
```

```
            // Conecta ao servidor especificado na URL
            //                                  Exemplo:                                  http://api.open-
meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current_weather=true
            http.begin("http://api.open-
meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current_weather=true");

            Serial.println("[HTTP] Fazendo requisição GET...");
            int httpCode = http.GET(); // Envia a requisição GET

            if (httpCode > 0) { // HTTP code será negativo em caso de erro
              Serial.printf("[HTTP] Código da Resposta: %d\n", httpCode);

              if (httpCode == HTTP_CODE_OK || httpCode == HTTP_CODE_MOVED_PERMANENTLY)
{
                String payload = http.getString(); // Obtém o corpo da resposta
                Serial.println(payload);
                // Aqui  processa-se o JSON recebido
              }
            } else {
              Serial.printf("[HTTP] Erro na requisição GET: %s\n", http.errorToString(httpCode).c_str());
            }
            http.end(); // Fecha a conexão
          }
          delay(10000); // Aguarda 10 segundos antes da próxima requisição
```

Fazendo Requisições POST: Enviando Dados para um Servidor

O método POST é utilizado para enviar dados para um servidor, geralmente para criar um recurso ou submeter informações. Pense em seu dispositivo como alguém preenchendo um formulário e enviando-o. Os dados a serem enviados (o "payload") são incluídos no corpo da requisição.

Exemplo Prático (Conceitual para HTTPClient com ESP32/ESP8266):

Um dispositivo coleta dados de um sensor e precisa enviá-los para um servidor central ou banco de dados.

```
// ... (cabeçalhos e conexão WiFi de setup) ...

void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // A URL onde se vai enviar os dados POST
    http.begin("http://seu_servidor.com/api/dados_sensor");

    // Definir o cabeçalho Content-Type para JSON
    http.addHeader("Content-Type", "application/json");

    // Dados a serem enviados (payload em formato JSON)
    float valorSensor = 23.45; // Leitura de um sensor real
    String jsonPayload = "{\"sensor_id\": \"temp_01\", \"valor\": " + String(valorSensor) + "}";

    Serial.print("[HTTP] Enviando POST com payload: ");
    Serial.println(jsonPayload);

    int httpCode = http.POST(jsonPayload); // Envia a requisição POST com o payload

    if (httpCode > 0) {
      Serial.printf("[HTTP] Código da Resposta POST: %d\n", httpCode);
      if (httpCode == HTTP_CODE_OK) { // Ou HTTP_CODE_CREATED (201)
        String response = http.getString();
        Serial.println("Resposta do servidor:");
        Serial.println(response);
      }
```

```
        } else {
          Serial.printf("[HTTP] Erro na requisição POST: %s\n",
http.errorToString(httpCode).c_str());
        }
        http.end();
      }
      delay(15000); // Aguarda 15 segundos
    }
```

Trabalhando com JSON (Serialização e Deserialização)

Na IoT, JSON (JavaScript Object Notation) é o formato de dados mais popular para troca de informações via HTTP devido à sua leveza e facilidade de leitura e escrita tanto por humanos quanto por máquinas.

- **Serialização:** É o processo de converter dados estruturados (como leituras de sensores) do seu firmware para uma string JSON que pode ser enviada (POST). Bibliotecas como ArduinoJson facilitam muito essa tarefa.
- **Deserialização:** É o processo de converter uma string JSON recebida (GET) de volta para dados estruturados que seu firmware pode entender e usar. ArduinoJson também é excelente para isso.

Dominar o uso de JSON é crucial para interagir com a maioria das APIs e serviços web.

5.2. Desenvolvendo um Servidor Web no Firmware (Opcional/Avançado)

Além de ser um cliente, seu microcontrolador pode se transformar em um servidor web, permitindo que outros dispositivos (como navegadores de um computador ou smartphone) se conectem diretamente a ele para obter informações ou enviar comandos. Isso é extremamente útil para interfaces de configuração local, monitoramento ou controle direto sem a necessidade de uma nuvem intermediária.

Cenários de Uso

- **Configuração Local:** Um navegador pode se conectar ao dispositivo para configurar credenciais de Wi-Fi, parâmetros de sensor, ou modos de operação.
- **Interface de Controle:** Ligar/desligar LEDs, acionar relés ou ajustar configurações através de botões ou sliders em uma página HTML servida pelo próprio dispositivo.
- **Monitoramento em Tempo Real:** Exibir leituras de sensores em gráficos simples diretamente no navegador.

Criando Páginas Web Simples

Servir páginas HTML dinâmicas diretamente do seu microcontrolador é um recurso poderoso. Pode-se embarcar strings HTML no seu código, concatená-las com dados em tempo real (como leituras de sensores), ou até mesmo carregar arquivos HTML/CSS/JS de um sistema de arquivos como o SPIFFS/LittleFS no ESP32/ESP8266. Isso permite criar interfaces de usuário mais ricas sem a necessidade de um aplicativo externo.

5.3. Boas Práticas e Segurança em HTTP no Firmware: Garantindo Robustez e Proteção

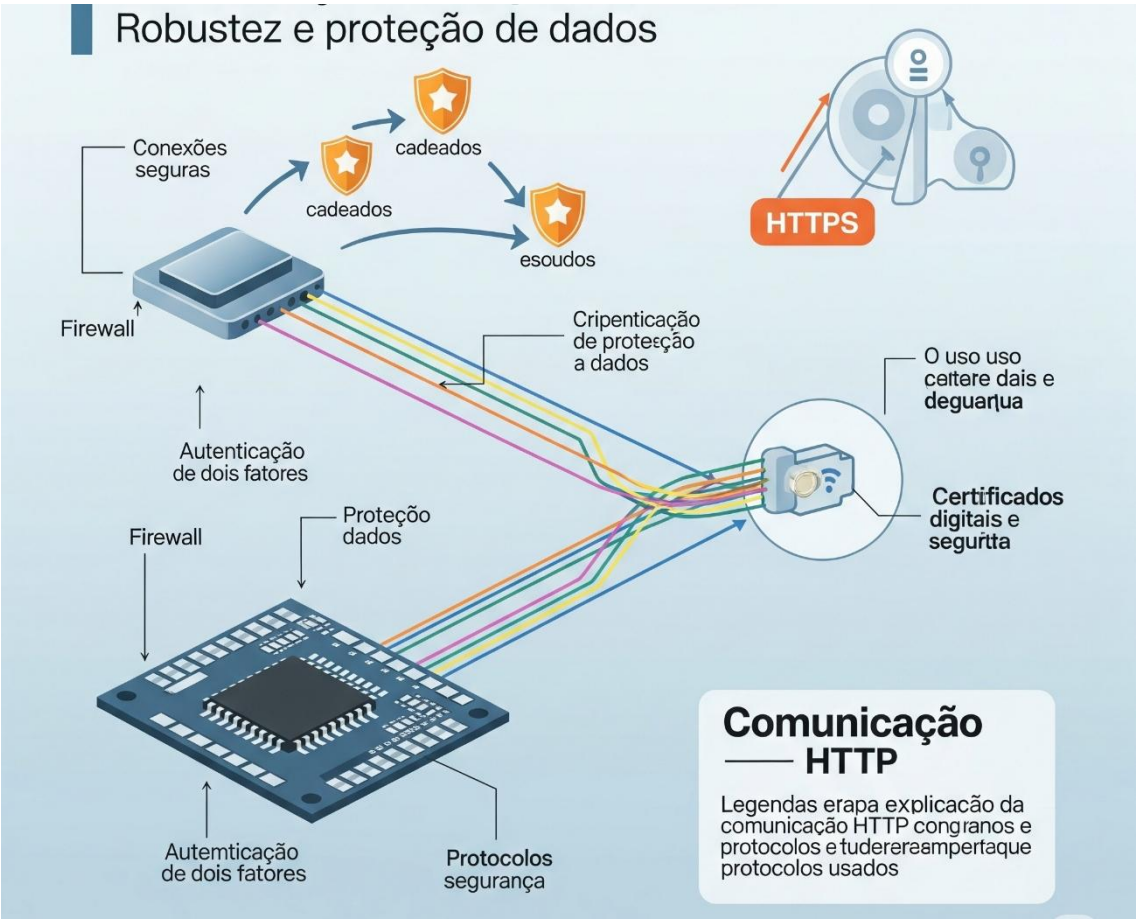
Ao integrar a comunicação HTTP no seu firmware, a robustez e a segurança são tão cruciais quanto a funcionalidade. Microcontroladores têm recursos limitados e geralmente operam em ambientes menos controlados, o que exige atenção extra a esses detalhes.

Para garantir que seu dispositivo se comunique de forma confiável e segura, siga estas boas práticas:

- **Tratamento de Erros Eficaz:** Sempre verifique o status da conexão Wi-Fi/Ethernet e os códigos de status HTTP retornados (como 200 OK, 404 Not Found, 500 Internal Server Error). Implemente lógica de retentativa (retry) para falhas temporárias e defina timeouts para evitar que o dispositivo "congele" esperando por respostas que nunca chegam. Isso torna seu firmware mais resiliente a problemas de rede ou servidor.
- **Priorize HTTPS para Segurança:** Para qualquer troca de dados sensíveis (senhas, informações pessoais) ou comandos críticos, o uso de HTTPS é mandatório. Essa camada de segurança (SSL/TLS) criptografa a comunicação, protegendo seus dados contra interceptação. Embora consuma mais recursos do microcontrolador, a segurança não é opcional em muitas aplicações de IoT.
- **Otimização de Recursos:** Microcontroladores não são PCs. Seja eficiente no uso de memória e processamento:
    - **Minimize o Payload:** Envie apenas os dados essenciais para economizar largura de banda e processamento.
    - **Gerencie a Memória:** Evite criar muitas strings temporárias e buffers grandes.
    - **Frequência de Requisições:** Envie dados apenas quando necessário, não em excesso, para preservar bateria e evitar sobrecarga na rede e no servidor.

Ao incorporar essas práticas, garante-se que seus dispositivos embarcados não só se comunicam via HTTP, mas o fazem de maneira confiável, eficiente e segura, prontos para os desafios do mundo real da IoT.

Figura 10: Boas Práticas e Segurança em Comunicação HTTP para Firmware



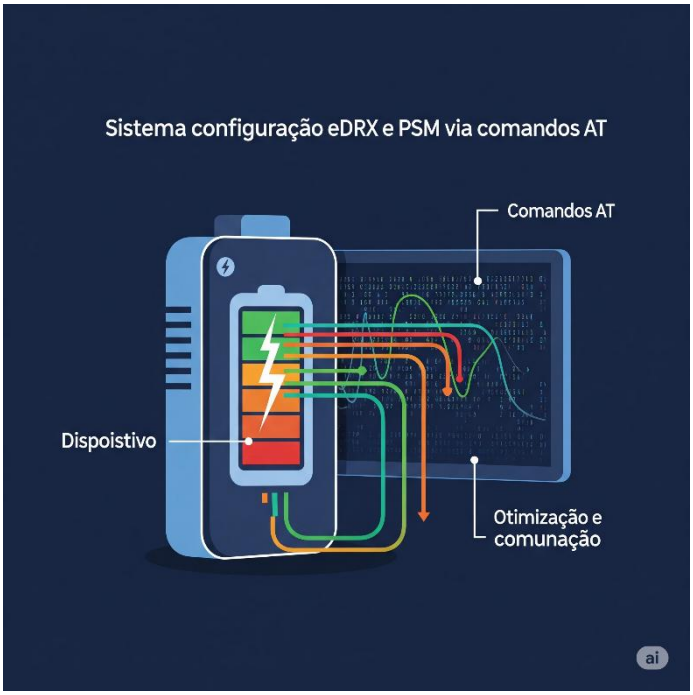
Fonte: Lab 4.1 (Equipe técnica do projeto)

Capítulo 6: Otimizando a Energia: Configuração eDRX e PSM via Comando AT

Para dispositivos IoT que operam com baterias e necessitam de uma vida útil prolongada, a eficiência energética é um fator crítico. É aqui que entram duas tecnologias fundamentais: eDRX (extended Discontinuous Reception) e PSM (Power Saving Mode). Ambas permitem que seu módulo de comunicação, geralmente um modem celular (como NB-IoT ou LTE-M), economize energia drasticamente, permanecendo "dormindo" por longos períodos e acordando apenas quando necessário.

Neste capítulo, será focado em como se podem configurar e gerenciar essas funcionalidades poderosas diretamente do firmware do microcontrolador, utilizando os ubíquos Comandos AT. Entender-se-á o eDRX e o PSM e saber-se-á como ativá-los e monitorá-los via comando AT, o que é essencial para maximizar a autonomia dos dispositivos IoT.

Figura 11: Otimizando a Energia: Configuração eDRX e PSM via Comando AT



Fonte: Lab 4.1 (Equipe técnica do projeto)

6.1. Fundamentos de eDRX e PSM para Economia de Energia

Neste tópico, exploraremos os conceitos por trás de eDRX e PSM, e como eles se complementam para otimizar o consumo de energia em módulos de comunicação celular.

- O Desafio da Energia em IoT Celular: Por que a gestão de energia é crucial para dispositivos conectados via redes móveis.
- Modo de Recepção Descontínua Estendida (eDRX):
  - O que é? Entenda como o eDRX permite que o módulo de comunicação "durma" por períodos mais longos, acordando em intervalos pré-definidos para verificar a existência de mensagens.
  - Ciclos de eDRX: A importância de configurar o ciclo de tempo de inatividade e atividade para equilibrar latência e economia.
  - Casos de Uso: Cenários onde o eDRX é mais vantajoso (e.g., monitoramento ocasional, atualizações de firmware).
- Modo de Economia de Energia (PSM):
  - O que é? Compreenda como o PSM leva a economia de energia a um nível ainda maior, permitindo que o módulo desligue completamente a sua parte de rádio, ficando em um estado de "sono profundo".
  - Tempo de Atividade (TAU - Tracking Area Update) e Tempo Ativo Periódico (Active Time): Como esses parâmetros definem o ciclo de vida do PSM.
  - Vantagens e Desvantagens: Comparativo rápido de PSM versus eDRX.
- Sinergia e Complementaridade: Como eDRX e PSM podem ser usados em conjunto para maximizar a vida útil da bateria.

6.2. Configurando e Monitorando eDRX e PSM via Comandos AT

Este tópico mergulhará na parte prática, mostrando como interagir diretamente com o módulo de comunicação usando Comandos AT para configurar e verificar o status de eDRX e PSM.

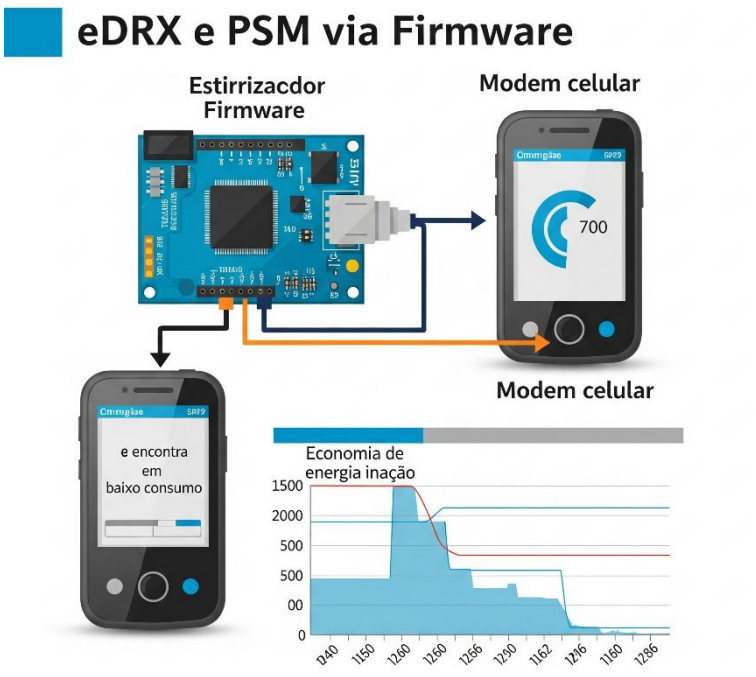
- Introdução aos Comandos AT: Uma breve revisão sobre o que são Comandos AT e como eles são a interface padrão para controlar modems celulares.
- Comandos AT Essenciais para eDRX:
    - Ativação/Desativação: AT+CEDRXS (Setting eDRX parameters).
    - Verificação de Status: AT+CEDRXP (Checking eDRX parameters).
    - Interpretação da Resposta: Como entender os valores retornados pelo módulo (ex: tempo de ciclo, valor real concedido pela rede).
- Comandos AT Essenciais para PSM:
    - Ativação/Desativação: AT+CPSMS (Setting PSM parameters).
    - Configuração de Timers: Definindo o "Requested Periodic TAU" e o "Requested Active Time".
    - Verificação de Status: AT+CEREG ou comandos específicos do fabricante para o estado de registro na rede, que pode indicar o PSM.
- Interação Firmware-Módulo:
    - Envio de Comandos AT: Como seu microcontrolador envia strings de comando AT via porta serial (UART) para o modem.
    - Parsing de Respostas: Como o firmware deve interpretar as respostas do modem para verificar o sucesso ou falha da configuração.
- Considerações Práticas:
    - Dependência da Rede: Entenda que a ativação e os parâmetros reais de eDRX/PSM podem ser limitados pela configuração da rede da operadora.
    - Testes e Monitoramento: A importância de testar as configurações em campo e monitorar o consumo real de energia.

Capítulo 7: O Controle Total: eDRX e PSM via Firmware

No capítulo anterior, foram desvendados os fundamentos de eDRX e PSM e foi aprendida a interação com esses recursos através de Comandos AT. Agora, será dado um passo além: será aprendida a implementação do controle total dessas tecnologias de economia de energia diretamente no coração do firmware.

Este capítulo é a ponte entre o entendimento conceitual e a aplicação prática e robusta. Será descoberto como o microcontrolador pode, de forma inteligente, gerenciar os modos de sono profundo e recepção estendida do módulo de comunicação, adaptando-se às necessidades do aplicativo e maximizando a vida útil da bateria dos dispositivos IoT.

Figura 12: eDRX e PSM via Comando AT



Fonte: Lab 4.1 (Equipe técnica do projeto)

7.1 Integração e Gerenciamento do Modem para eDRX e PSM

Este tópico aborda como o firmware do seu microcontrolador se comunica e gerencia as funcionalidades de economia de energia do modem.

- Comunicação Firmware-Modem:
  - Revisão da interface serial (UART) para envio e recebimento de Comandos AT.
  - Bibliotecas e abstrações comuns para comunicação UART em microcontroladores (Ex: Serial no Arduino, ou drivers de UART específicos do SDK).
- Sequência de Inicialização e Configuração:
  - Como o firmware deve inicializar o modem e a rede antes de configurar eDRX/PSM.
  - Ordem recomendada para o envio de comandos AT de configuração (Ex: configurar PSM antes de eDRX, se aplicável).
- Gerenciamento Dinâmico de Modos de Energia:
  - Quando e como o firmware deve ativar ou desativar eDRX/PSM com base nas condições da aplicação (Ex: envio de dados agendado, detecção de eventos).
  - Estratégias para transição entre modos de baixo consumo e modos de atividade plena.

Figura 13: Diagrama de Blocos Integração e Gerenciamento do Modem para eDRX e PSM

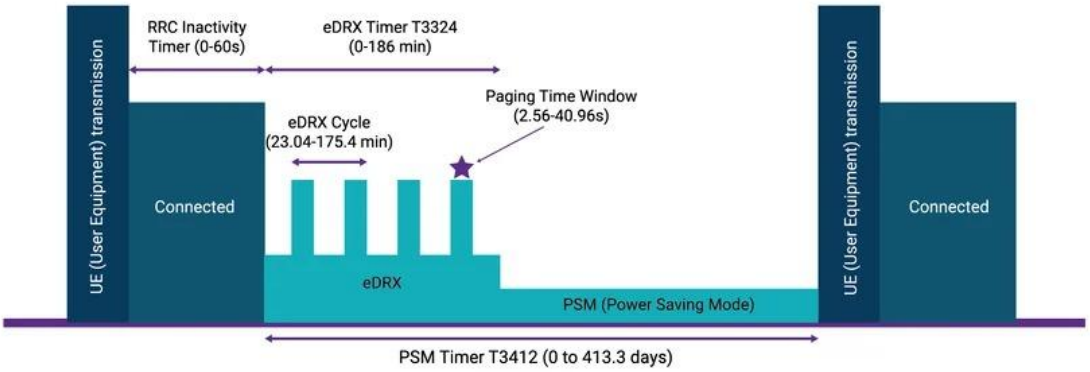


Diagram 1: Power Saving Mode (PSM) and Extended Discontinuous Reception (eDRX) mechanisms in NarrowBand IoT (NB-IoT)

Fonte: https://blog.velosiot.com/what-are-psm-edrx-features-in-lte-m-and-nb-iot(2022)

7.2 Implementação de Rotinas de eDRX e PSM no Firmware

Aqui o foco será os exemplos de código e na lógica necessária para controlar eDRX e PSM de forma eficaz.

- **Configurando eDRX via Firmware:**
  - Exemplo de função em C/C++ para enviar o comando AT+CEDRXS e interpretar a resposta do modem.
  - Lógica para definir os parâmetros de Requested_eDRX_value (P-T-W - Paging Time Window, e ciclos de eDRX).
  - Tratamento de **respostas da rede** (rede pode conceder um valor diferente do solicitado).
- **Configurando PSM via Firmware:**
  - Exemplo de função em C/C++ para enviar o comando AT+CPSMS e interpretar a resposta.
  - Definição de Requested_Periodic_TAU e Requested_Active_Time.
  - Considerações sobre a **persistência da configuração** entre reinícios do módulo.
- **Sincronização e Despertar (Wake-up):**
  - Como o firmware pode sincronizar suas tarefas com os períodos de atividade do modem.
  - Estratégias para forçar o modem a "acordar" para tarefas urgentes (Ex: recebimento de comando prioritário).
- **Monitoramento e Depuração:**
  - Técnicas para monitorar o status de eDRX/PSM e o consumo de energia através do firmware.
  - Uso de logs seriais para depurar a comunicação e a configuração dos modos de energia.

Referências

ESPRESSIF. Documentação oficial da biblioteca WiFi do ESP32 (Arduino). [S. l.], [2020]. Disponível em: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi. Acesso em: 28 jul. 2025.

ESPRESSIF. Documentação oficial do ESP32. [S. l.], [2020]. Disponível em: https://docs.espressif.com. Acesso em: 28 jul. 2025.

MQTT.ORG. Site oficial do protocolo. [S. l.], [2020]. Disponível em: https://mqtt.org. Acesso em: 28 jul. 2025.

RANDOM NERD TUTORIALS. Tutorial: conectando o ESP32 ao Wi-Fi e realizando uma requisição HTTP. [S. l.], [2020]. Disponível em: https://randomnerdtutorials.com/esp32-http-get-post-arduino/. Acesso em: 28 jul. 2025.

RANDOM NERD TUTORIALS. Tutorial prático MQTT com ESP32 + Mosquitto + MQTT.fx. [S. l.], [20--?]. Disponível em: https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/. Acesso em: 28 jul. 2025.

SCIKIT-LEARN. Documentação do Scikit-learn: biblioteca de machine learning em Python. [S. l.], [2020]. Disponível em: https://scikit-learn.org/stable/. Acesso em: 28 jul. 2025.

THE THINGS NETWORK. Comunidade e plataforma aberta para LoRaWAN. [S. l.], [2020]. Disponível em: https://www.thethingsnetwork.org. Acesso em: 28 jul. 2025.