JOHANNES KEPLER
UNIVERSITY LINZ

Author
Michael Reinegger, BSc
12102754

Submission
Project in Networks and Security (353.093)

May 5, 2026

Analysis of Power Consumption of Home and SME Network Devices

Seminar Report

Master Project about the Analysis of Power Consumption of different Home to SME Network Equipment.

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Contents

Acronyms

API Application Programming Interface

ARP Address Resolution Protocol

CPE Customer Premises Equipment

CPU Central Processing Unit

CSV Comma-Separated Values

DECT Digital Enhanced Cordless Telecommunications

DHCP Dynamic Host Configuration Protocol

DUT Device Under Test

EEE Energy-Efficient Ethernet

GbE Gigabit Ethernet

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

ISP Internet Service Provider

LAN Local Area Network

MAC Media Access Control

NIC Network Interface Card

NFC Near Field Communication

ONT Optical Network Terminal

PHY Physical Layer

SOAP Simple Object Access Protocol

SSE Server-Sent Events

SME Small and Medium-sized Enterprise

TCP Transmission Control Protocol

UDP User Datagram Protocol

USB Universal Serial Bus

WLAN Wireless Local Area Network

Abstract

This report investigates the real world power consumption of four home and Small and Medium-sized Enterprise (SME) network devices, comparing measurements against manufacturer specifications. The devices under test are the `Fritzbox 7530`, `Asus RT-AX68U`, `Huawei EG8245W5-8T`, and `Alcatel HH40V`.

To measure this properly, I built a custom Go-based web application that generates customizable network load while logging power consumption from a Digital Enhanced Cordless Telecommunications (DECT) 200 smart plug via the FritzBox TR-064 Application Programming Interface (API). Five tests cover incremental Ethernet port loading, per-port throughput scaling, full subsystem profiling (Ethernet + Wi-Fi + USB combined), Wi-Fi distance and band comparison, and bridge-mode efficiency.

Key findings: devices only reach 22–47% of their provided power supply rated maximum power under full combined load, with my tests. Wi-Fi radios are the dominant power consumer when active, drawing 7–8× more power per megabit than wired Ethernet. Running an ISP modem in bridge mode alongside a dedicated router increases combined idle power by about 86% compared to using the modem standalone. Built-in eco modes have varying effectiveness: the Fritzbox's eco mode saves 7–19% but limits Ethernet throughput to ~100 Mbps/port, while Wi-Fi TX power reduction on the Asus and Huawei yields minimal savings.

1. Introduction

Consumer to SME network routers operate continuously, yet manufacturer specifications typically only state maximum power ratings (e.g., 18 W or 33 W). This report aims to quantify the baseline and load-dependent power consumption of these devices in typical configurations.

The following four devices were tested: a `Fritzbox 7530` (ISP-style router, 2018), an `Asus RT-AX68U` (Wi-Fi 6, 2020), a `Huawei EchoLife EG8245W5-8T` (fiber Optical Network Terminal (ONT) from Energie AG), and an `Alcatel HH40V` (basic single-band, 2018). Stated power ratings range from 10–33 W. To evaluate real-world power efficiency, power draw was empirically measured across these devices to determine how power consumption changes with Ethernet port count, Wi-Fi load, eco mode, or running in bridge mode.

To do this I built a custom Go application that generates controlled User Datagram Protocol (UDP) traffic on each Ethernet interface, polls the DECT 200 smart plug via TR-064 [11], and stores everything in a SQLite database. The source code is at `github.com/MrCodeEU/jku-master-project-network-devices-power-consumption-analysis`.

A few problems occurred along the way: the smart plug only updates every two minutes by default (worked around via Home Assistant), one Network Interface Card (NIC) port was broken, and some routers had quirks resulting in deviations from the planned protocol – all covered in the relevant sections.

2. Research Questions

These questions were originally phrased differently but naturally changed while testing and running the experiments. The final versions are:

RQ1 How does power consumption change with the number of active Ethernet ports?

RQ2 What is the per-port power overhead of idle Ethernet connections, and how does it compare to the incremental cost of active network traffic?

RQ3 How does power consumption scale with network load (idle, 50%, 100% utilization of bandwidth)?

RQ4 Which equipment is suitable for continuous operation and which devices have high idle consumption that makes them candidates for automated shutdown during the night?

RQ5 How does Wi-Fi affect router power consumption, and how do factors such as frequency band (2.4 vs. 5 GHz), number of connected clients, and signal strength (distance to access point) influence it?

RQ6 What is the actual impact of manufacturer-provided power-saving modes (e.g., Fritzbox `Green Mode`, Wi-Fi transmission power control) on overall power consumption?

RQ7 Does a device's maximum power consumption rating correlate with its real-world power usage under typical operating conditions, and what does it take to reach that level?

RQ8 How efficient is ISP-provided equipment (e.g., Huawei router or Fritzbox 7530) when operating in bridge mode compared to dedicated networking equipment?

3. Setup

3.1 Hardware

The four devices under test are routers and modems I had available – a mix of ISP-provided equipment and personally-owned hardware:

Fritzbox 7530 (2018, 802.11ac, 4× Gigabit Ethernet (GbE), rated 18 W max [3]) – ISP-style ADSL/VDSL router, also used as the dedicated power-data collection device

Asus RT-AX68U (2020, 802.11ax, 4× GbE, rated 33 W max [2]) – high-end consumer Wi-Fi 6 router

Huawei EchoLife EG8245W5-8T (2020, 802.11ax, 4× GbE, rated 24 W max [10]) – fiber/GPON ONT modem provided by Energie AG

Alcatel HH40V (2018, 802.11n, 2× 100 Mbps, rated 10 W max [1]) – basic budget ISP modem

3.2 Sampling Rate and Accuracy of Power Meter

The DECT 200 accuracy is rated at ±100 mW below 5 W and ±2% above 5 W [4], giving ~70 mW effective resolution as per my testing which is sufficient for this project.

By default the plug only updates its reading every two minutes. Longer phases are therefore needed for meaningful averages – which is why Test 1 is significantly longer than the others. The workaround (found after the first test): the DECT 200 refreshes whenever it receives any command. A Home Assistant automation sends `turn on` every 15 seconds, and the test app polls via TR-064 every 5 seconds, effectively getting a fresh reading every ~15 s [9].

```
alias: Automation - Nudge FritzDECT Smart Plug
description: ""
triggers:
  - trigger: time_pattern
    seconds: /15
conditions:
  - condition: state
    entity_id: switch.fritz_dect_200
    state: "on"
  - condition: time
    after: "06:30:00"
    before: "23:59:00"
actions:
  - action: switch.turn_on
    target:
```

```
16        entity_id: switch.fritz_dect_200
17  mode: single
```

3.3 Initial Testing Setup

The setup (Figure 1) uses a dedicated FritzBox 7490 as a "data collector" that is never under test itself – it hosts the DECT 200 base station and provides the TR-064 API, while the Device Under Test (DUT) is plugged into the smart plug. This separation is critical: under heavy UDP load the DUT's management API becomes unresponsive, causing measurement gaps when the same device is used for both.

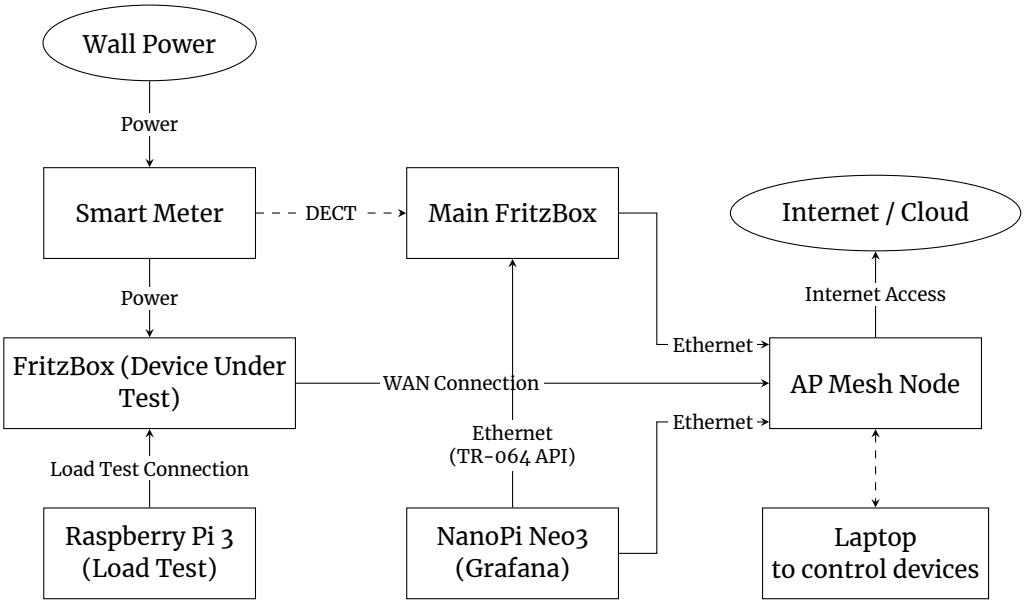


Figure 1: Initial Network Layout

Figure 2 shows an early Grafana dashboard from the first exploratory measurements. The power jump during the iperf run is already visible; the periodic spikes later are most likely internal FritzBox background activity.

3.4 Getting Network Connectivity Right

The initial testing setup involved a Raspberry Pi 3 as shown in Figure 1, which due to the gigabit Ethernet port being connected to the internal USB controller is not sufficiently fast to generate enough load. Therefore, I used a different PC as test PC that has a 4-port 2.5 GbE NIC, but port 4 turned out to be broken (links up, drops all frames). The workaround was three NIC ports plus the motherboard Intel I225-V port.

Each DUT uses a different default subnet, so the test PC interfaces needed manual IP configuration before each run. Example for the Fritzbox (192.168.188.x):

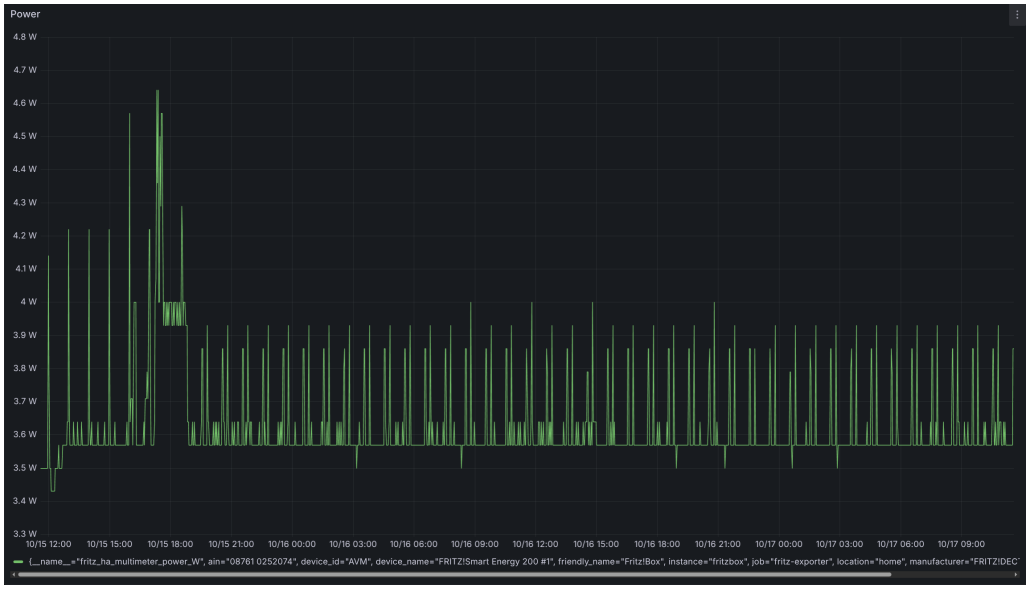


Figure 2: Early Grafana dashboard showing FritzBox power consumption during an initial iperf test.

```powershell
# Remove old addresses and assign FritzBox subnet IPs
Remove-NetIPAddress -InterfaceAlias "Ethernet 5" -Confirm:$false
    -ErrorAction SilentlyContinue
New-NetIPAddress -InterfaceAlias "Ethernet 5" -IPAddress 192.168.188.3
     -PrefixLength 16
New-NetIPAddress -InterfaceAlias "Ethernet 7" -IPAddress 192.168.188.4
     -PrefixLength 16
New-NetIPAddress -InterfaceAlias "Ethernet"   -IPAddress 192.168.188.5
     -PrefixLength 16
New-NetIPAddress -InterfaceAlias "Ethernet 6" -IPAddress 192.168.188.6
     -PrefixLength 16
```

Subnets per device: FritzBox 192.168.188.x, Huawei 192.168.18.x, Asus 192.168.51.x, Alcatel 192.168.1.x. Static host routes were added so each interface reached the DUT's management IP without relying on Windows' routing:

```powershell
# Routes to Huawei gateway, different metric per interface
New-NetRoute -DestinationPrefix "192.168.18.1/32" -InterfaceAlias "
    Ethernet 5" `
    -NextHop "0.0.0.0" -RouteMetric 2
New-NetRoute -DestinationPrefix "192.168.18.1/32" -InterfaceAlias "
    Ethernet 7" `
    -NextHop "0.0.0.0" -RouteMetric 3
```

Connectivity was verified with `ping` and `Test-NetConnection` before each run via the test app's web UI.

3.5 Automating Testing and Data Collection

To make measurements reproducible, I built a custom web application with a Go backend and JavaScript frontend. Source at `github.com/MrCodeEU/jku-master-project-network-devices-power-consumption-analysis`.

3.5.1 Architecture Overview

The application consists of a few cooperating components:

Test Runner (`runner.go`) Orchestrates three phases (pre-test baseline, load with per-interface ramping, post-test baseline). Durations, interface order, and ramp steps are all configurable from the web UI.

Load Generator (`loadgen.go`) Generates UDP flooding, Transmission Control Protocol (TCP), or raw L2 frames. Per-interface rate limiting uses nanosecond-precision spin-wait. UDP flooding was the mode used for all Ethernet tests.

Power Meter Interface (`fritzbox.go`) Reads power from the DECT 200 via TR-064 Simple Object Access Protocol (SOAP) [11] using `gofritz` [5]. Always talks to the dedicated collection FritzBox, never the DUT.

Web Server (`server.go`) REST API + Server-Sent Events (SSE) for real-time browser streaming. Results persisted in SQLite.

Analysis Dashboard (`analysis.js`) Computes per-phase stats (avg, std dev, min/max, Mbps/W) and renders Chart.js visualizations. Exports Comma-Separated Values (CSV), Excel, and pgfplots LaTeX [8] – the latter is how some of the figures in this report were generated.

3.5.2 Load Generation Approach

Three methods were tried to generate network load from the test PC to the router: TCP, UDP, and raw Layer 2 frames. A custom Go application handles all of them. The choice of protocol and implementation had significant effects on the maximum throughput.

TCP was evaluated using standard Go `net.DialTCP` connections. The test application creates multiple concurrent TCP connections targeted at the router's management IP, continuously writing random data payloads into the sockets.

```
1  func (g *NetworkLoadGenerator) runTCPWorkerWithConfig(
2      ctx context.Context, id int, config Config, ic InterfaceConfig) {
3      targetAddr, _ := net.ResolveTCPAddr("tcp", fmt.Sprintf("%s:%d",
       config.TargetIP, config.TargetPort))
4      localAddr, _ := g.getLocalAddr(ic.Name, "tcp")
5
6      // Explicitly bind the local NIC to bypass OS routing
7      dialer := &net.Dialer{Timeout: 5 * time.Second, LocalAddr:
       localAddr}
8      conn, _ := dialer.DialContext(ctx, "tcp", targetAddr.String())
9      defer conn.Close()
10
```

```
    if tcpConn, ok := conn.(*net.TCPConn); ok {
        tcpConn.SetNoDelay(true) // Disable Nagle's algorithm for max throughput
        tcpConn.SetWriteBuffer(4 * 1024 * 1024)
    }

    buffer := make([]byte, config.PacketSize)
    rand.Read(buffer) // Prevent payload compression

    for {
        // ... rate limiting and writing ...
        conn.Write(buffer)
    }
}
```

However, TCP topped out at merely 60–80 Mbps total throughput. This is presumably because the routers proactively rate-limit unsolicited TCP connections or rapidly aborts connections directed to their own management IPs or closed ports but also on the open web interface ports.

Raw L2 frames via `libpcap` (using a Go wrapper) bypass the system's IP stack entirely, allowing the generator to directly dump pre-constructed Ethernet frames onto the wire. To maximize throughput, the generator pre-serializes a static payload and writes directly to an inactive `pcap` handle activated in immediate mode, avoiding OS routing entirely.

```
func (lg *NetworkLoadGenerator) layer2Worker(
    ctx context.Context, ifaceConfig InterfaceConfig,
    srcMAC, dstMAC net.HardwareAddr, handle *pcap.Handle, payloadSize int) {

    // Pre-serialize packet to avoid per-packet processing overhead
    ethLayer := &layers.Ethernet{
        SrcMAC:       srcMAC,
        DstMAC:       dstMAC,
        EthernetType: layers.EthernetTypeIPv4,
    }
    buffer := gopacket.NewSerializeBuffer()
    gopacket.SerializeLayers(buffer, opts, ethLayer, gopacket.Payload(payload))
    packetData := buffer.Bytes()

    // Optimization: Send packets in tight loops (bursts)
    const burstSize = 128
    for {
        for i := 0; i < burstSize; i++ {
            handle.WritePacketData(packetData) // Dump frame straight to wire
        }
        // ... apply rate limiting delay and context checks only between bursts
    }
}
```

While this could theoretically hit wire-speed, it was capped at ~550 Mbps on Windows. The bottleneck is likely in OS scheduling overhead and the user-space to kernel-space transition per packet. While this could potentially be improved with kernel-bypass libraries (like DPDK), they add too much complexity and I am not good enough at C to implement something like that and call it via Go or rather CGo.

UDP flooding, in the end, proved to be the most simple and well enough working method. Using standard Go `net.DialUDP` but batching the writes, it successfully bypassed the TCP handshake issues and the `libpcap` software bottlenecks.

```
func (g *NetworkLoadGenerator) runUDPWorkerWithConfig(
    ctx context.Context, id int, config Config, ic InterfaceConfig) {
    targetAddr, _ := net.ResolveUDPAddr("udp", fmt.Sprintf("%s:%d",
    config.TargetIP, config.TargetPort))

    // Crucial: Binding our outbound UDP socket specifically to one
    physical NIC.
    localAddr, _ := g.getLocalAddr(ic.Name, "udp")
    conn, _ := net.DialUDP("udp", localAddr.(*net.UDPAddr), targetAddr
    )
    defer conn.Close()

    conn.SetWriteBuffer(4 * 1024 * 1024) // 4MB buffer offsets OS
    jitter

    // ... setup buffer and begin infinite write loop ...
}
```

UDP flooding performed the best, achieving near line-rate (e.g., 3.85 Gbps aggregate inbound throughput on the Huawei across 4×1 GbE ports) using 1400-byte packets, 16 concurrent workers per interface, and 10-packet batches. Because of this, UDP flooding was chosen as the sole generation method for all subsequent Ethernet load tests.

Methodology & Assumptions. All generated UDP traffic purposefully targets the DUT's own IP address rather than routing *through* it to an external network. Because the targeted UDP port is closed, we must assume that the packets are initially processed by the device's physical network interface, passed to the OS network stack, and subsequently dropped. This forced protocol evaluation reliably generates a highly intensive Central Processing Unit (CPU) interrupt load that stresses the device's processing capabilities far more aggressively than pure hardware Layer 2 switching would. While it is not a perfect emulation of typical consumer web traffic, it forces substantial power consumption and gives us a repeatable performance ceiling.

Implementation Details & Code Level Optimization. To ensure multiple outgoing Local Area Network (LAN) connections are utilized simultaneously, the Go load-generator explicitly binds a dedicated socket to each local physical network adapter before dialing the DUT IP inside an infinite loop. This bypasses

the Windows routing table which would otherwise funnel all traffic out of a single default interface. A simplified snippet of the worker payload loop is shown below:

```
func (g *NetworkLoadGenerator) runUDPWorkerWithConfig(
    ctx context.Context, id int, config Config, ic InterfaceConfig) {

    // Resolve target and explicitly bind local socket to the specific
     NIC
    targetAddr, _ := net.ResolveUDPAddr("udp", config.TargetIP)
    localAddr, _ := g.getLocalAddr(ic.Name, "udp")
    conn, _ := net.DialUDP("udp", localAddr.(*net.UDPAddr), targetAddr
    )
    defer conn.Close()

    conn.SetWriteBuffer(4 * 1024 * 1024)
    buffer := make([]byte, config.PacketSize) // 1400 byte frame
    rand.Read(buffer) // Pseudo-randomized payload

    const batchSize = 10
    packetCount := 0
    delay := g.getWorkerDelayForInterface(config.PacketSize, ic.Name)

    for {
        select {
        case <-ctx.Done(): return
        default:
            n, _ := conn.Write(buffer) // Dispatch flood
            g.updateInterfaceThroughput(ic.Name, n)
            packetCount++

            // Batching sleep calls reduces scheduling overhead
            if delay > 0 && packetCount >= batchSize {
                PreciseSleep(delay * time.Duration(batchSize))
                packetCount = 0
                // Re-poll the desired throughput limit (dynamic
    ramping)
                delay = g.getWorkerDelayForInterface(config.PacketSize
    , ic.Name)
            }
        }
    }
}
```

A critical hurdle in achieving smooth partial-load bandwidth steps (e.g., scaling smoothly from 250 Mbps up to 1000 Mbps in Test 2) was the lack of timer precision in the Windows OS kernel. Standard `time.Sleep()` calls in Go typically default to a 10–15 millisecond resolution window. When rate-limiting nanosecond-scale packets, this low resolution caused wildly fluctuating data spikes rather than a flat, consistent throughput. To overcome this limitation, our `PreciseSleep` interface leverages a hybrid timing approach: utilizing specific Windows high-resolution waitable timers (`CreateWaitableTimerExW`) from the `kernel32.dll` to capture the bulk of the milliseconds, immediately followed by CPU spin-locking for the final microsecond tolerance limit.

In addition to the custom UDP/TCP/Layer 2 load generator, **iperf3** [6] was used for Wi-Fi throughput testing. The load generation tool integrates iperf3 as a subprocess, parsing its per-second output to record throughput alongside power data. For phone-to-phone Wi-Fi load tests (Test 3), an Android iperf3 app [12] was used to generate traffic between two smartphones connected to the DUT's Wi-Fi network. For Wi-Fi-to-LAN tests (Test 4), the iperf3 server ran on the test PC while a Wi-Fi client device acted as the iperf3 client, routing traffic through the DUT.

4. Test Setups

This section describes the five test setups, covering DUT, topology, traffic generation, and protocol. Practical deviations are noted where relevant.

4.1 Test 1: Incremental Port Load Test (Completed)

Incrementally loads 1–4 Ethernet ports with maximum UDP traffic, directly addressing **RQ1** and partially **RQ4** and **RQ7**.

4.1.1 Devices Under Test

All four devices (as described in Section 3): `Fritzbox 7530`, `Asus RT-AX68U`, `Huawei EG8245W5-8T`, `Alcatel HH40V`.

4.1.2 Physical Topology

The PC connects 1–4 Ethernet cables to the DUT, which is powered through the DECT 200 smart plug. A separate `Fritzbox 7490` (not under test) hosts the DECT 200 base station and TR-064 API – see Figure 3.

4.1.3 Test Protocol

Each device was tested using the following automated protocol, executed by the custom web application described in Section 3.5:

1. **Pre-test baseline** (10 min): All Ethernet cables connected but no traffic generated. Measures idle power with active link-up on all ports.
2. **Load phase 1** (20 min): Maximum UDP flood on **1 port** (target: 1 Gbps).
3. **Load phase 2** (20 min): Maximum UDP flood on **2 ports** (target: 2 Gbps).
4. **Load phase 3** (20 min): Maximum UDP flood on **3 ports** (target: 3 Gbps). *Skipped for `Alcatel HH40V` (only 2 ports).*
5. **Load phase 4** (20 min): Maximum UDP flood on **4 ports** (target: 4 Gbps). *Skipped for Alcatel.*

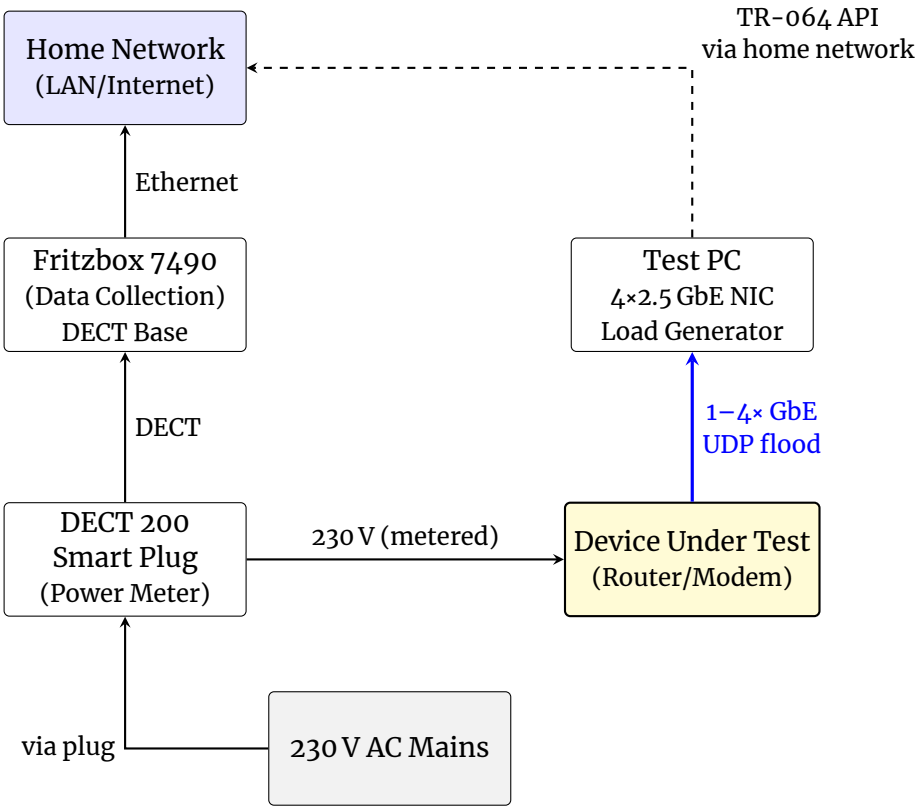


Figure 3: Physical topology for the Incremental Port Load Test.

6. **Post-test baseline** (20 min): All traffic stopped, cables remain connected. Measures power recovery behavior.

Load generation parameters:

- Protocol: UDP, packet size: 1400 bytes, 16 workers per interface
- Target: device's own IP address (e.g., `169.254.1.1:80` for Fritzbox). The UDP flood is directed at closed ports on the DUT. The packets are processed by the device's network interface and dropped by the OS network stack, which still successfully generates the intended CPU interrupt load without requiring a dedicated listening service.
- Interfaces activated incrementally with 5 ramp-up steps per interface
- Power polling interval: 5 seconds (via TR-064 SOAP API)

Important limitation: Traffic is directed at the DUT's own IP address, meaning all packets are processed by the device's *CPU/software stack*. This represents a CPU-intensive workload (similar to, but likely much higher than, heavy NAT processing or connection handling) rather than pure L2 forwarding. As a result, devices with weaker CPUs (notably the `Fritzbox 7530`) show throughput degradation at higher port counts due to CPU saturation, not switching fabric limitations.

4.2 Test 2: Throughput Scaling (Single Port)

Single-port throughput ramp from 0 to line rate, directly addressing **RQ2** and **RQ3**. Comparing single-cable idle with Test 1's four-cable idle quantifies per-port overhead.

4.2.1 Devices Under Test

All four devices from Test 1. GbE devices target 1000 Mbps; Alcatel targets 100 Mbps (its native link speed).

4.2.2 Physical Topology

Identical to Test 1 (Figure 3) but with a single Ethernet cable. The throughput ramp profile is shown in Figure 4.

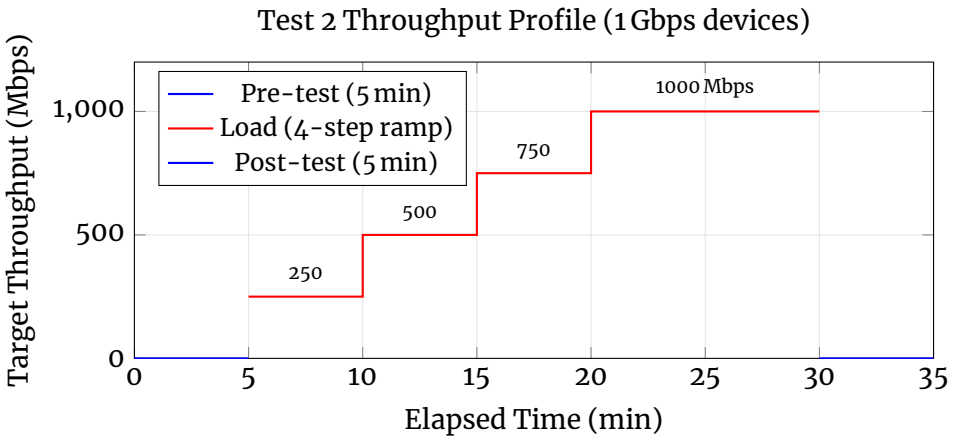


Figure 4: Throughput profile for Test 2. The load generator's ramp-step feature automatically transitions through four equal-duration steps at 25%, 50%, 75%, and 100% of the link capacity. For the Alcatel HH40V, steps are at 25, 50, 75, and 100 Mbps.

4.2.3 Test Protocol

The test uses the load generator's built-in *ramp steps* feature, which increases the per-interface rate limit in equal increments:

1. **Pre-test baseline** (5 min): 1 cable connected, no traffic.
2. **Load phase** (25 min total): Single port, UDP, `ramp_steps=4`, `target_mbps=1000`. The load generator automatically steps through:
   - Step 1: ~250 Mbps (25% of link capacity)
   - Step 2: ~500 Mbps (50%)
   - Step 3: ~750 Mbps (75%)
   - Step 4: ~1000 Mbps (100%)

3. **Post-test baseline** (5 min): Traffic stopped, cable remains connected.

Load generation parameters:

- Protocol: UDP, packet size: 1400 bytes, 10 workers
- Single interface enabled, `ramp_duration` = 0 (auto, equals load duration)
- Target: DUT's own IP address (consistent with Test 1)
- Power polling interval: 5 seconds

Important limitation: Traffic is directed at the DUT's own IP address, meaning all packets are processed by the device's *CPU/software stack*. Since it is on an closed UDP port the packets are received and then dropped. This represents a CPU-intensive workload (similar to, but likely much higher than, heavy NAT processing or connection handling) rather than pure L2 forwarding. As a result, devices with weaker CPUs (notably the `Fritzbox 7530`) show throughput degradation at higher port counts due to CPU saturation, not switching fabric limitations.

4.3 Test 3: Incremental Subsystem Power Profiling

Progressively enables every subsystem from bare power-on to full combined load. Six groups are recorded as separate runs and combined for analysis. Two runs per device: Run A (defaults) and Run B (manufacturer power-saving modes). Addresses **RQ2**, **RQ4**, **RQ5**, **RQ6**, and **RQ7**.

4.3.1 Devices Under Test

All four devices. The Alcatel (2×100 Mbps, 2.4 GHz only, no USB) skips groups it is not capable of.

4.3.2 Physical Topology

Same as Test 1. Two smartphones act as Wi-Fi clients (one per band), for Groups 5–6, an Android iperf3 app generates phone-to-phone Wi-Fi traffic through the DUT.

4.3.3 Group Design

Six groups cumulatively add subsystems (Figure 5). Individual cable additions are retained because Test 1 starts with all cables connected and cannot isolate the per-cable Physical Layer (PHY) cost.

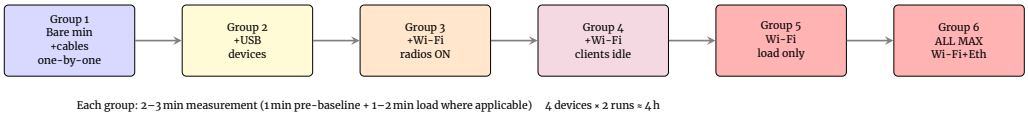


Figure 5: Test 3 group sequence.

4.3.4 Test Protocol — Run A (Default Settings)

Subsystems are added *cumulatively*. Each group is a separate run (1 min pre-baseline, 1–2 min measurement at 5 s polling); manual intervention between groups modifies the physical setup. This is also the reason why the tests are split into multiple groups instead of a single run with all transitions, to keep the measurement phases clean and comparable across devices.

Group 1 Device on, all Wi-Fi off, no cables. Record bare minimum, then add cables *one at a time* (~2 min each) with custom events marking each insertion.

Group 2 Attach USB storage devices one at a time. *Skipped for Alcatel (no USB).*

Group 3 Enable Wi-Fi radios: 2.4 GHz first, then 5 GHz after ~2 min (no clients). *Alcatel: 2.4 GHz only. Fritzbox: both bands enable together.*

Group 4 Connect smartphones: one to 2.4 GHz, one to 5 GHz (idle, no traffic).

Group 5 iperf3 flood between the two phones via Android app; tool records power only.

Group 6 Wi-Fi iperf3 flood continues + simultaneous max UDP on all Ethernet ports.

4.3.5 Test Protocol — Run B (Power-Saving Modes)

Run B repeats Groups 3–6 with manufacturer-provided power-saving modes enabled. Since power-saving features primarily affect Wi-Fi transmission power and (for some devices) Ethernet PHY settings, Groups 1–2 (bare minimum, cables, USB) are not repeated; the Run A data is reused for those groups. The following power-saving settings were applied:

- **Fritzbox 7530:** Full eco mode enabled (System → Energy Monitor → `Green Mode`), which reduces Ethernet PHY power, Wi-Fi transmission power, and dims LEDs. The Fritzbox eco mode was the only one to affect Ethernet, so a *full* rerun of all groups (1–6) was performed.
- **Asus RT-AX68U:** Wi-Fi TX power set to the lowest available setting for both bands.
- **Huawei EG8245W5-8T:** Wi-Fi TX power set to 20% for both bands.
- **Alcatel HH40V:** No power-saving features available; Run B was skipped.

4.3.6 Measurement Considerations

Because each group is recorded as a separate test run, minor power discontinuities may appear when combining the data across groups. In practice, the observed jumps were small (typically <100 mW) and certainly do not affect the analysis.

4.3.7 Research Questions Addressed

RQ2: Group 1 isolates the per-port PHY link-up cost by adding cables individually, complementing the per-port load data from Test 1.

RQ4: The bare-minimum measurement in Group 1 provides the absolute power floor.

RQ5: Groups 3–5 isolate the idle cost of Wi-Fi radios, client associations, and active Wi-Fi load. Individual-band throughput measurements are covered in detail by Test 4.

RQ6: Direct comparison of Run A vs. Run B at each group quantifies savings from manufacturer eco/power-saving modes.

RQ7: Group 6 loads all subsystems simultaneously, providing the closest approximation to the rated maximum power.

4.3.8 Practical Notes and Challenges

Device-specific notes:

- **Fritzbox bands**: Both 2.4/5 GHz radios switch on/off together – impossible to isolate per-band cost in Group 3.
- **Fritzbox port anomaly**: One port ("Ethernet 7") reported ~25 000 Mbps during Group 6 – the NIC was counting retransmitted frames. The other three ports showed normal ~461 Mbps each; Group 6 throughput was estimated by multiplying that average by four. Power data was unaffected.
- **Power-saving modes**: Alcatel has none – Run B skipped. Fritzbox eco mode also throttles Ethernet PHY to ~100 Mbps, so a full six-group rerun was done for it rather than just Groups 3–6.

4.4 Test 4: Wi-Fi Signal Strength and Distance

This test measures the effect of frequency band (2.4 GHz vs. 5 GHz), physical distance, and effect of walls on Wi-Fi power consumption and throughput. It addresses **RQ5**.

4.4.1 Devices Under Test

All four devices are tested:

- **Asus RT-AX68U** (802.11ax, dual-band 2.4 + 5 GHz, 160 MHz capable)
- **Fritzbox 7530** (802.11ac, dual-band 2.4 + 5 GHz)
- **Huawei EG8245W5-8T** (802.11ax, dual-band 2.4 + 5 GHz)
- **Alcatel HH40V** (802.11n, 2.4 GHz only)

4.4.2 Physical Setup

- **Near position:** Client device <2 m from DUT, line of sight.
- **Wall position:** Client device ~10 m from DUT, through one interior wall.

The DECT 200 measures DUT power. For the Asus, Fritzbox, and Alcatel, a Wi-Fi client device runs iperf3 to the test PC connected via Ethernet LAN. The load generator tool records both power and throughput inline.

4.4.3 Test Protocol

Each scenario (device × band × distance) is a separate 5-minute run: 1 min idle baseline, 3 min iperf3 TCP flood, 1 min post-test baseline. Dual-band devices are tested four times (2 bands × 2 distances); the Alcatel is tested twice (2 distances on 2.4 GHz only).

4.4.4 Device-Specific Deviations

- Fritzbox: Both radios always active. The `Fritzbox 7530` firmware does not allow disabling individual radio bands. Both the 2.4 GHz and 5 GHz radios remain active during all tests. The idle baseline therefore includes the power draw of both radios, and tests labeled "5 GHz" or "2.4 GHz" reflect which band carries the iperf3 traffic while the other radio is idle.
- Huawei: Wi-Fi client isolation. The Huawei EG8245W5-8T isolates LAN and Wi-Fi clients by default (and I was not able to find a way to turn it off), preventing iperf3 from running between a wired PC and a Wi-Fi client. Throughput is captured separately from the iperf3 terminal output (1-second intervals, averaged to 5-second bins for analysis).
- Alcatel: 100 Mbps LAN bottleneck. The Alcatel has only two 100 Mbps Ethernet ports, creating a potential throughput bottleneck. To verify, additional "both Wi-Fi" tests were performed with the test PC also connected via Wi-Fi, bypassing the LAN entirely.

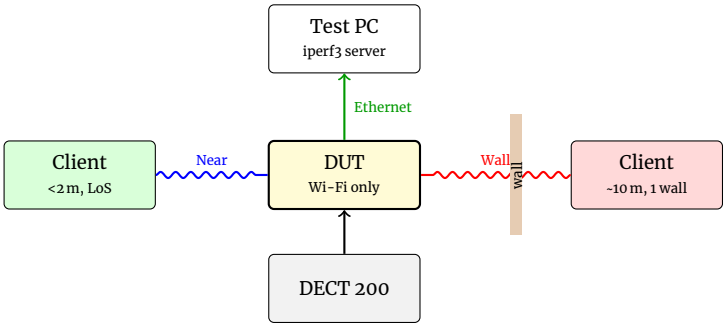


Figure 6: Test 4 setup for Wi-Fi distance measurements.

4.4.5 Research Questions Addressed

RQ5: Comparing near vs. wall power for the same band isolates the effect of signal attenuation on DUT power consumption. Comparing 2.4 GHz vs. 5 GHz at the same distance reveals band-dependent power and throughput characteristics. Cross-device comparison quantifies how router hardware and Wi-Fi standard (802.11n/ac/ax) affect energy efficiency.

4.5 Test 5: ISP Bridge Mode Comparison

This test investigates whether splitting ISP-provided equipment into bridge mode plus a dedicated router reduces total system power consumption compared to using the ISP device in standalone router mode. It directly addresses **RQ8**.

4.5.1 Devices Under Test

Configuration A (Standalone): Huawei EG8245W5-8T operating as router (NAT, DHCP, Wi-Fi, firewall). This configuration reuses the Huawei results already obtained in Test 1 (Section 4.1), since the protocol and load parameters are the same.

Configuration B (Split): Huawei EG8245W5-8T in bridge mode (minimal processing) + Asus RT-AX68U as dedicated router.

4.5.2 Physical Topology

Figure 7 compares the two configurations.

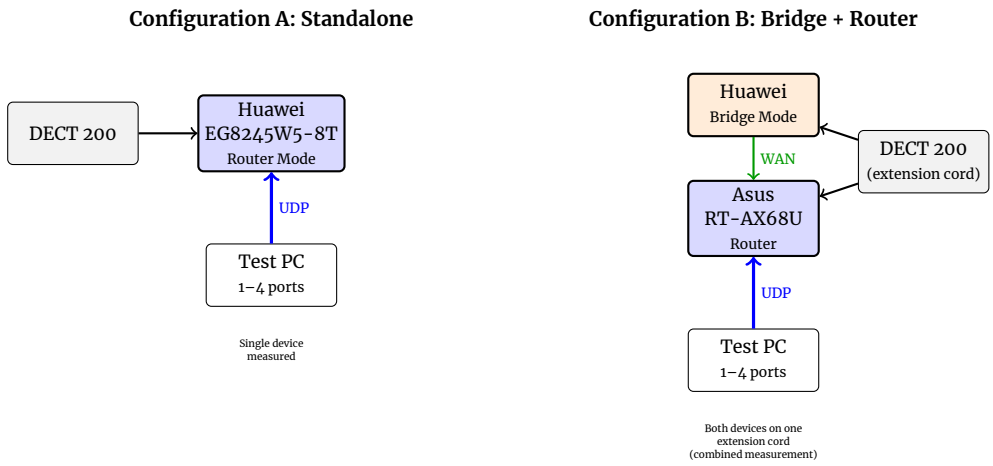


Figure 7: Test 5 topology comparison.

4.5.3 Test Protocol

Both configurations are tested with the same incremental port load profile as Test 1 for direct comparability:

1. **Pre-test baseline** (10 min): Cables connected, no traffic.
2. **Load phases** (4×20 min): Incremental 1–4 port UDP flood.
3. **Post-test baseline** (10 min): Traffic stopped.

Configuration A: Reuses the Huawei standalone measurements from Test 1 (Section 5). Because the load profile, traffic parameters, and power metering setup are identical, no additional measurement run is needed; the Huawei Test 1 data serves directly as the single-device baseline.

Configuration B:

- Huawei set to bridge mode via its admin interface (disables NAT, DHCP, firewall, optionally Wi-Fi).
- Asus RT-AX68U configured as the primary router (WAN port connected to Huawei's LAN port).
- Test PC connects to the Asus's LAN ports.
- **Power measurement:** Both devices on a single DECT 200 via a power strip to get the combined power consumption.

Load generation parameters: Identical to Test 1 (UDP, 1400 bytes, 16 workers, 5 s polling).

4.5.4 Practical Notes

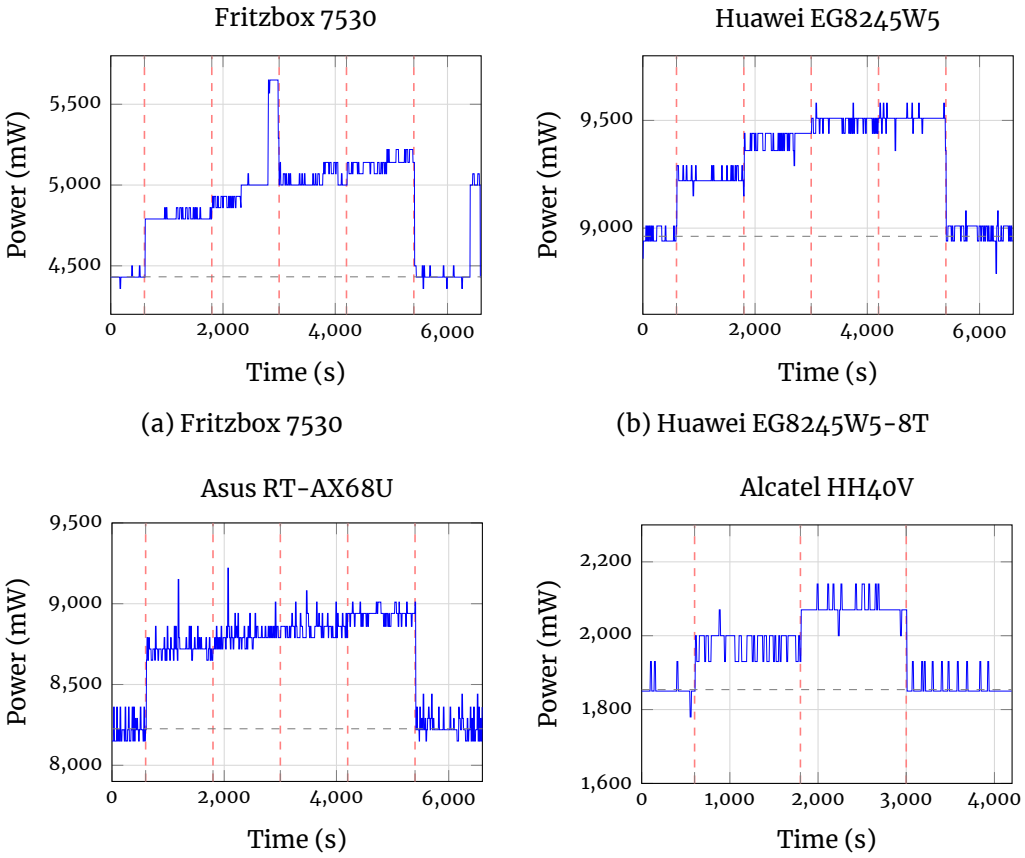
Reaching the Huawei's management IP (192.168.18.1) from the test PC (in the Asus's 192.168.51.x subnet) required two routing changes:

1. On the **Asus router**: add a static route sending 192.168.18.0/24 traffic out via the WAN interface.
2. On the **test PC**: add a persistent route pointing to the Huawei's specific IP via the Asus's LAN IP:

```
# On Windows test PC: route to Huawei's management IP via the Asus
    router
route -p add 192.168.18.1 mask 255.255.255.255 192.168.51.1 metric 1
```

5. Results: Incremental Port Load Test

This section presents the results of Test 1 (Section 4.1), which incrementally loads 1–4 Ethernet ports with maximum UDP traffic on all four devices.



(a) Fritzbox 7530

(b) Huawei EG8245W5-8T

(c) Asus RT-AX68U

(d) Alcatel HH40V (2 ports only)

Figure 8: Test 1 power consumption over time for all DUTs.

5.1 Power Time-Series

Figure 8 shows the power consumption over time for all four devices. Dashed vertical lines mark phase transitions and horizontal dashed lines indicate the idle baseline.

The `Fritzbox 7530` hit a CPU bottleneck: the aggregate inbound throughput across all connected ports actually *decreased* from an aggregate ~2483 Mbps (across 3 ports) and ~1827 Mbps at 4 ports as the CPU saturated, while power stayed flat around 5000–5130 mW. More ports, less throughput but same power consumption. The Huawei and Asus both hit an aggregate inbound throughput of ~3815 Mbps across all 4 separate GbE ports (which have a theoretical maximum of 4 Gbps combined) with clean monotonic power increases of only +548 mW and +707 mW respectively. The Alcatel (2×100 Mbps) topped out at 257 Mbps with a +222 mW increase.

5.2 Cross-Device Comparison and RQ Analysis

Table 1 summarises the test findings as well as some additional observations and annual costs with an average power price of 0.25 € per kWh [7].

Table 1: Test 1 summary across all four devices.

Metric	Fritzbox	Huawei	Asus	Alcatel
Idle Power (mW)	4 432	8 962	8 226	1 854
Max Load Power (mW)	5 132	9 510	8 933	2 076
Power Increase (mW / %)	+700 (15.8)	+548 (6.1)	+707 (8.6)	+222 (12.0)
Max Throughput (Mbps)	2 483	3 815	3 816	257
Rated Max (W) / Utilisation	18 / 31%	24 / 40%	33 / 28%	10 / 21%
Annual Idle Cost (€)	9.70	19.63	18.01	4.06

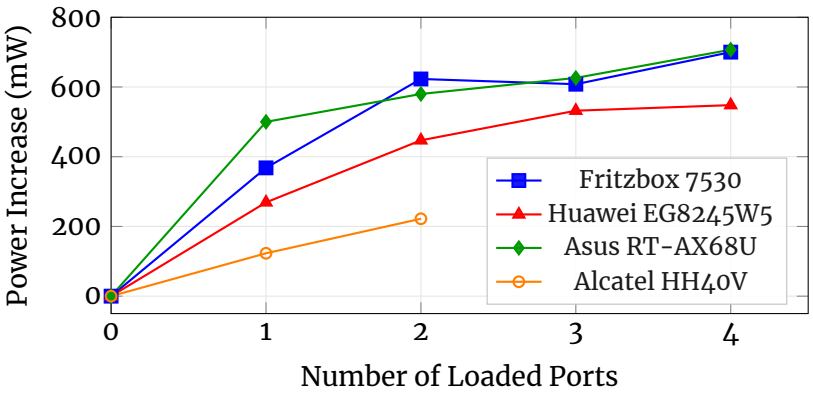


Figure 9: Power increase relative to Link-Up Idle as 0 mW.

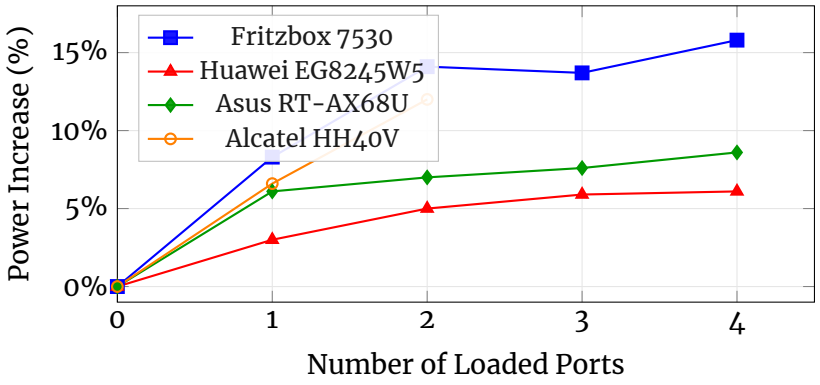


Figure 10: Percentage increase compared to Link-Up Idle.

RQ1: Port count vs. power. It is important to differentiate between **Base Idle** (no cables connected) and **Link-Up Idle** (cables connected, active link, but no traffic). In this section, "idle" refers to Link-Up Idle, which already incurs a baseline power cost. Relative to Link-Up Idle, per-port power increases from traffic load are surprisingly small at just 30–175 mW per port. As Figure 10 shows, the relative increase across all active ports never exceeds 16% of the Link-Up Idle baseline across tested devices. In fact, adding load on already connected ports costs significantly less than the static Link-Up Idle cost of having the cables plugged in (see Section 6.3). This distinction is further confirmed when evaluating the energy efficiency (power increase relative to throughput): an additional 1 Mbps of traffic typically incurs only around 0.1–0.3 mW per Mbps of dynamic load on modern GbE devices (ignoring the static connection cost). The

Fritzbox shows a non-monotonic profile: power at 3 ports (5040 mW) is *lower* than at 2 ports (5055 mW) because CPU saturation means less actual work is done. The Huawei and Asus scale mostly monotonically as their CPUs are not the bottleneck at 4×1 Gbps.

RQ4 (partial): Idle power. Idle power ranges from 1.85 W (Alcatel) to 8.96 W (Huawei) – nearly a 5× spread between the cheapest and most power-hungry device. Annual idle costs are €4–€20 at €0.25/kWh [7]. Note: these measurements have cables connected; bare-minimum power (no cables, Wi-Fi off) is covered in Test 3.

RQ7 (partial): Rated vs. real maximum. With only Ethernet active, devices reach 21–40% of their rated maximum power. The spec sheet numbers have a lot of headroom. Adding Wi-Fi, USB, and other subsystems closes the gap further but never fully. See Test 3 for the full picture.

6. Results: Throughput Scaling (Single Port)

Test 2 uses a single Ethernet cable and ramps throughput in four steps (250/500/750/1000 Mbps for GbE devices; 25/50/75/100 Mbps for the Alcatel). Having only one cable connected makes it easy to compare with Test 1's four-cable idle and figure out the actual per-port overhead (RQ2).

6.1 Power Time-Series

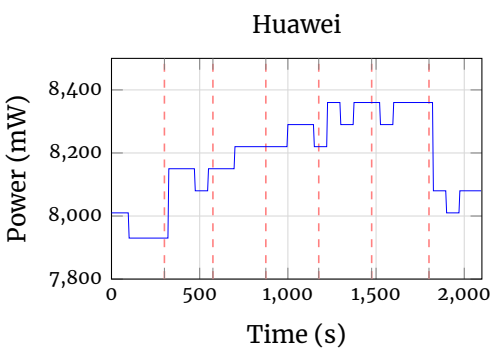
6.2 Per-Step Statistics

The Huawei and Asus both scale sub-linearly – each 250 Mbps step costs less power than the previous one. The Asus hits an interesting plateau: the 750→1000 Mbps step adds only +4 mW. The Fritzbox looks strange here because the pre-test idle was elevated from the previous test run. The post-test value (3 584 mW) is closer to the real baseline. Also its power actually *decreases* at high load due to CPU saturation, as mentioned before. The Alcatel (100 Mbps) is the odd one out, scaling linearly: each 25 Mbps step adds ~30–46 mW, for a total of +120 mW (+7.1%) at full load.

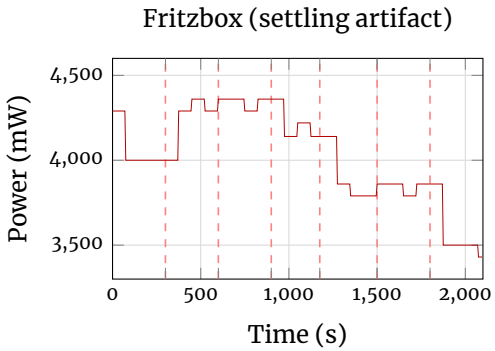
6.3 Per-Port Idle Overhead (RQ2)

Comparing single-cable idle from this test with four-cable idle from Test 1 gives us the per-port overhead directly:

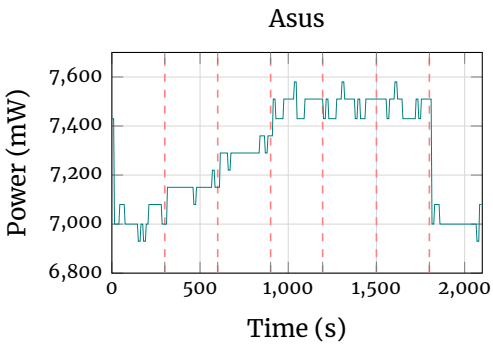
The measurements have been verified across different physical cables and ports with a variance of 5–10% to rule out localized hardware defects. The Fritzbox has the most power-efficient GbE ports at ~120 mW each; the Asus costs the most at ~395 mW/port. More strikingly: for the Asus and Huawei, three idle cables together (1.0–1.2 W) cost *more* than running a single port at full 1 Gbps load (~0.4 W). The cable just being plugged in is more expensive than the data flowing through it.



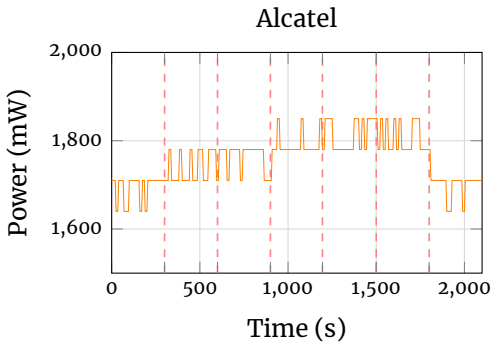
(a) Huawei: 7 957→8 344 mW (+4.9%)



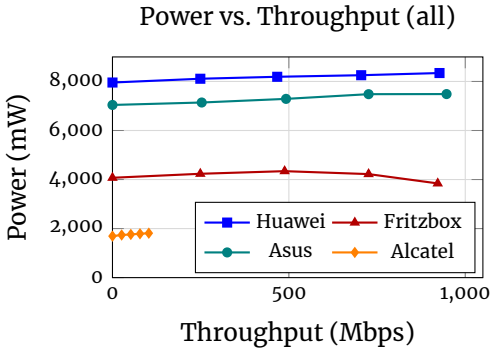
(b) Fritzbox: power *decreases* at high load



(c) Asus: 7 042→7 486 mW (+6.3%)



(d) Alcatel: 1 692→1 812 mW (+7.1%)



(e) Power vs. Throughput for all devices.

Figure 11: Test 2 single-port ramp results.

Table 2: Test 2: Per-step power and throughput for all devices (single port).

		Power (mW)		Throughput		
Device	**Phase**	**Avg**	σ	**Avg (Mbps)**	**Target**	**Eff. (Mbps/W)**
Huawei	Idle	7 957	38	—	—	—
	250 Mbps	8 111	66	249	250	30.7
	500 Mbps	8 191	35	467	500	57.0
	750 Mbps	8 255	35	705	750	85.4
	1000 Mbps	8 344	30	927	1000	111.1
	Post	8 086	89	—	—	—
Fritzbox	Idle	4 072	127	—	—	—
	250 Mbps	4 235	140	249	250	58.8
	500 Mbps	4 342	31	488	500	112.5
	750 Mbps	4 222	92	726	750	172.0
	1000 Mbps	3 842	31	922	1000	239.9
	Post	3 584	162	—	—	—
Asus	Idle	7 042	102	—	—	—
	250 Mbps	7 142	40	253	250	35.4
	500 Mbps	7 290	45	492	500	67.5
	750 Mbps	7 482	51	726	750	97.1
	1000 Mbps	7 486	44	947	1000	126.5
	Post	7 034	115	—	—	—
Alcatel	Idle	1 692	—	—	—	—
	25 Mbps	1 738	—	26	25	15.0
	50 Mbps	1 762	—	52	50	29.5
	75 Mbps	1 788	—	78	75	43.6
	100 Mbps	1 812	—	103	100	56.8
	Post	1 640	—	—	—	—

Table 3: Per-port idle overhead: Test 1 (4 cables) vs. Test 2 (1 cable). Per-port = $\Delta/3$ for GbE; direct difference for Alcatel.

Device	**4-cable (mW)**	**1-cable (mW)**	**Δ (mW)**	**Per-Port (mW)**
Fritzbox 7530	4 432	4 072	360	~120
Huawei EG8245W5	8 962	7 957	1 005	~335
Asus RT-AX68U	8 226	7 042	1 184	~395
Alcatel HH40V	1 854	1 692	162	~162

7. Results: Incremental Subsystem Power Profiling

Test 3 is the most comprehensive one: it starts with each device powered on and nothing else connected, then adds subsystems one by one (Ethernet cables, USB, Wi-Fi radios, clients, load) to see exactly what each component costs. Run A used factory defaults; Run B repeated the Wi-Fi groups with manufacturer power-saving modes enabled. Except for the Alcatel, which has no such options.

7.1 Per-Device Subsystem Power Profiles

Table 4 shows the cumulative power at each subsystem stage for all devices (Run A, default settings).

Table 4: Test 3 Run A (default settings): Mean power at each cumulative subsystem stage. "Δ" columns show the incremental power added by that subsystem. All values in mW.

Subsystem Stage	Asus		Huawei		Fritzbox		Alcatel	
	Power	Δ	Power	Δ	Power	Δ	Power	Δ
Bare Minimum	3 576	—	6 342	—	1 983	—	1 140	—
+1st Cable	4 140	+564	6 629	+287	2 210	+227	1 361	+221
+2nd Cable	4 430	+290	6 874	+245	2 471	+261	1 484	+123
+3rd Cable	4 739	+309	7 173	+299	2 790	+319	—	—
+4th Cable	4 785	+46	7 430	+257	3 041	+251	—	—
+USB	5 227	+442	7 793	+363	3 460	+419	—	—
+WiFi 2.4 GHz	6 525	+1 298	8 580	+787	5 205	+1 745	1 903	+419
+WiFi 5 GHz	7 883	+1 358	9 240	+660			—	—
+WiFi Clients	7 986	+103	9 329	+89	5 338	+133	2 300	+397
WiFi Load	11 332	+3 346	10 573	+1 244	6 289	+951	2 205	−95
All MAX Load	12 286	+954	11 306	+733	6 794	+505	2 242	+37

Note: The cable/port power overhead is discussed in detail in Section 7.2.

Key observations. The Asus shows: the Wi-Fi radios alone cost 2 656 mW more than all four Ethernet ports put together. Active Wi-Fi load on top adds another 3 346 mW. That is a lot of power just for wireless. Which makes sense given the broader channel bandwidth. The Huawei starts high (6 342 mW bare minimum), likely because the GPON fiber transceiver is always running, there is no way to turn it off. Under full load it reaches 47.1% of it's rated 24 W which is the closest any device gets to its rated maximum power draw. The Fritzbox can't enable Wi-Fi bands one at a time. Therefore, both radios turn on together (+1 745 mW in one step). Under full load it hits 37.7% of rated 18 W. The Alcatel is remarkable for how little it draws: below 2.3 W even at full load. Its Wi-Fi load phase actually *reduced* power by 95 mW. It appears to not scale power consumption with throughput at all.

7.2 Per-Port PHY Link-Up Cost (RQ2)

Table 5 isolates the per-port PHY link-up cost (idle cable, no traffic).

Table 5: Per-port PHY link-up cost (mW) from Test 3 Group 1.

Device	+Cable 1	+Cable 2	+Cable 3	+Cable 4	Avg/port
Asus RT-AX68U	+564	+290	+309	+46	+302
Huawei EG8245W5	+287	+245	+299	+257	+272
Fritzbox 7530	+227	+261	+319	+251	+265
Alcatel HH40V	+221	+123	—	—	+172

These PHY link-up costs derived from Test 3 (adding cables one-by-one from zero) align somewhat closely with the top-down per-port derivations from Test 2's Table 3 (subtracting one cable from four). The Asus showed an average of ~395 mW/port in Test 2 vs. +302 mW/port here, while the Huawei showed ~335 mW/port vs. +272 mW/port here. The Fritzbox showed a notably lower cost in Test 2 (~120 mW) compared to its +265 mW average here, which highlights how non-linear the first-port initialization penalty can be compared to subsequent ports.

GbE port idle costs are consistent: 265–302 mW/cable. The Asus first cable costs +564 mW as the PHY probably does a heavier init going from zero to one active port. By cable 4 it adds only +46 mW, suggesting the first port carries most of the overhead. The Alcatel's 100 Mbps ports are more efficient at +172 mW each.

7.3 WiFi and USB Subsystem Costs (RQ4, RQ5)

Table 6 breaks down the marginal power cost per subsystem.

Table 6: Marginal subsystem power cost (mW). WiFi Load throughput from manual iperf3 observations.

Subsystem	Asus	Huawei	Fritzbox	Alcatel
All Ethernet (idle)	+1209	+1088	+1058	+344
USB	+442	+363	+419	—
WiFi Radios (idle)	+2656	+1447	+1745	+419
2.4 GHz only	+1298	+787	(both)	+419
5 GHz only	+1358	+660	(both)	—
WiFi Clients (idle)	+103	+89	+133	+397
WiFi Load (Δ)	+3346	+1244	+951	-95
at ~Mbps	147	58	194	17.5
Bare Min → Full Load	+8710	+4964	+4811	+1102

The Wi-Fi radio numbers are interesting: on the Asus, idle radios cost 2656 mW which is more than all four Ethernet cables combined (1209 mW). If you want to reduce a device's power consumption without changing hardware, turning off unused radio bands appears to be the most impactful. USB adds 363–442 mW which is more than I expected. The testing was done with a Kingston DataTraveler 100 G3 256GB USB 3.1 flash drive attached but not actively read/written to. Idle client association barely registers at 89–133 mW with the radio being on costing far more than phones being connected to it.

Power-Saving Modes (RQ6)

Run B repeated Groups 3–6 with manufacturer provided power-saving options turned on. Table 7 shows the difference.

Table 7: Run A vs. Run B (eco/power-saving mode). ΔEco: negative = saving. Fritzbox eco affects all subsystems; Asus/Huawei only WiFi TX power.

Device	Stage	Run A (mW)	Run B (mW)	ΔEco (mW)
Asus	WiFi 2.4 GHz ON	6 525	6 409	-116
	WiFi 5 GHz ON	7 883	7 734	-149
	WiFi Load	11 332	10 976	-356
	All MAX	12 286	12 251	-35
Huawei	WiFi 2.4 GHz ON	8 580	8 417	-163
	WiFi 5 GHz ON	9 240	9 177	-63
	WiFi Load	10 573	12 719	+2 146
	All MAX	11 306	13 478	+2 172
Fritzbox	Bare Min	1 983	1 930	-53
	4 Cables (idle)	3 041	2 475	-566
	+USB	3 460	2 920	-540
	WiFi Radios ON	5 205	4 535	-670
	WiFi Load	6 289	5 704	-585
	All MAX	6 794	6 320	-474

The **Fritzbox** eco mode is clearly the most useful of the three (**RQ6**): it saves 474–670 mW (7–19%) across all stages. The catch is it throttles the Ethernet PHY down to ~95 Mbps per port instead of the normal ~460 Mbps – so it is a real trade-off between energy and performance. The **Asus** TX power reduction is basically pointless: 35–356 mW saved (<3%). Not worth the bother. The **Huawei** result is confusing: idle Wi-Fi power dropped slightly as expected, but under load Run B drew +2 146 mW *more* than Run A. My best guess is that reducing TX power forced the radio into more retransmissions and lower modulation orders, which cost more power overall. Either way, treat the Huawei eco result with skepticism, but I did repeat that test and got similar results.

Approaching Rated Maximum Power (RQ7)

Table 8 shows Group 6 peak power vs. the spec sheet numbers.

Table 8: Test 3 maximum measured power vs. manufacturer rated maximum (**RQ7**).

Device	Measured Max (mW)	Rated Max (mW)	Utilisation
Asus RT-AX68U	12 286	33 000	37.2%
Huawei EG8245W5	11 306	24 000	47.1%
Fritzbox 7530	6 794	18 000	37.7%
Alcatel HH40V	2 242	10 000	22.4%

Not one device gets above 50% of its spec sheet maximum, even with every subsystem active and under full load. As far as my tests go at least. Compared to Test 1 (Ethernet only), adding Wi-Fi and USB bumped utilisation from 21–40% to 22–47%, but still nowhere near the rated limit. Manufacturer specs clearly include a lot of headroom for safety and reliability reasons, but it is interesting to see how much of that headroom is actually unused in real-world scenarios.

7.4 Summary of Findings

- **RQ2:** Each idle GbE cable costs 265–302 mW; Except for Alcatel: ~172 mW.
- **RQ4:** Bare-minimum power ranges from 1.14 W (Alcatel) to 6.34 W (Huawei). With everything connected but idle: 2.30–7.99 W, or €5–€18/year in electricity.
- **RQ5:** Wi-Fi radios are the biggest idle power draw (419–2 656 mW). Turning off unused bands is the single most impactful thing you can do. However, this also suggests an interesting architectural trade-off: since Wi-Fi radios incur a massive static power penalty just by being turned on, but physical Ethernet ports cost ~300 mW each just to maintain a link, a household could technically save total power by moving low-bandwidth devices (like smart home hubs or printers) to Wi-Fi *if* the Wi-Fi network is already active anyway, to reduce the PHY link-up cost of multiple Ethernet ports. But this needs further investigation to see how well it would work with multiple clients.
- **RQ6:** Fritzbox eco mode genuinely works – saves 474–670 mW (7–19%) at the cost of capping Ethernet to ~100 Mbps/port. Asus and Huawei TX power reduction: slight reduction to actually making it worse.
- **RQ7:** Even with everything on and at full load, no device exceeds 47% of its rated maximum.

8. Results: Wi-Fi Power and Throughput

Test 4 looks at Wi-Fi specifically: four routers, two bands (2.4/5 GHz), two positions (near: <2 m line of sight; wall: ~10 m through one interior wall). Each scenario is a 5-minute run: 1 min idle baseline, 3 min iperf3 TCP load, 1 min post. A few device quirks affected the results: The Fritzbox always runs both radios (firmware limitation). The Huawei isolates LAN and Wi-Fi clients so tests had to run phone-to-phone. The Alcatel is 2.4 GHz only, plus some extra runs were done with the test PC also on Wi-Fi to check whether its 100 Mbps LAN was bottlenecking things (it wasn't).

8.1 Power Consumption Overview

Table 9 has all the numbers: idle power, load power, the Wi-Fi overhead (ΔP = load - idle), throughput, and efficiency for every scenario.

Table 9: Test 4 summary: Wi-Fi power and throughput per device, band, and distance. ΔP denotes the increase from idle to load. Efficiency is computed as throughput / load power.

Device	Band	Dist.	Idle (mW)	Load (mW)	ΔP (mW)	Tput (Mbps)	Eff. (Mbps/W)
Asus	5 GHz	Near	5 860	8 852	+2 992	584	66.0
Asus	5 GHz	Wall	5 825	9 576	+3 751	244	25.5
Asus	2.4 GHz	Near	5 483	8 583	+3 101	143	16.6
Asus	2.4 GHz	Wall	5 448	6 443	+996	33	5.2
Fritzbox	5 GHz	Near	3 605	4 393	+788	235	53.5
Fritzbox	5 GHz	Wall	3 588	4 611	+1 023	61	13.2
Fritzbox	2.4 GHz	Near	3 553	4 078	+525	89	21.9
Fritzbox	2.4 GHz	Wall	3 930	4 303	+373	51	11.9
Huawei	5 GHz	Near	7 510	8 145	+635	83	10.2
Huawei	5 GHz	Wall	7 490	8 264	+774	36	4.4
Huawei	2.4 GHz	Near	7 267	8 723	+1 456	32	3.7
Huawei	2.4 GHz	Wall	7 278	8 813	+1 534	23	2.6
Alcatel	2.4 GHz	Near	1 780	2 579	+799	94	36.3
Alcatel	2.4 GHz	Wall	1 763	2 933	+1 171	88	30.1
Alcatel extra: test PC also on Wi-Fi (bypassing 100 Mbps LAN)							
Alcatel	2.4 GHz	Near*	1 588	2 211	+623	86	39.0
Alcatel	2.4 GHz	Wall*	1 624	2 390	+766	64	26.6

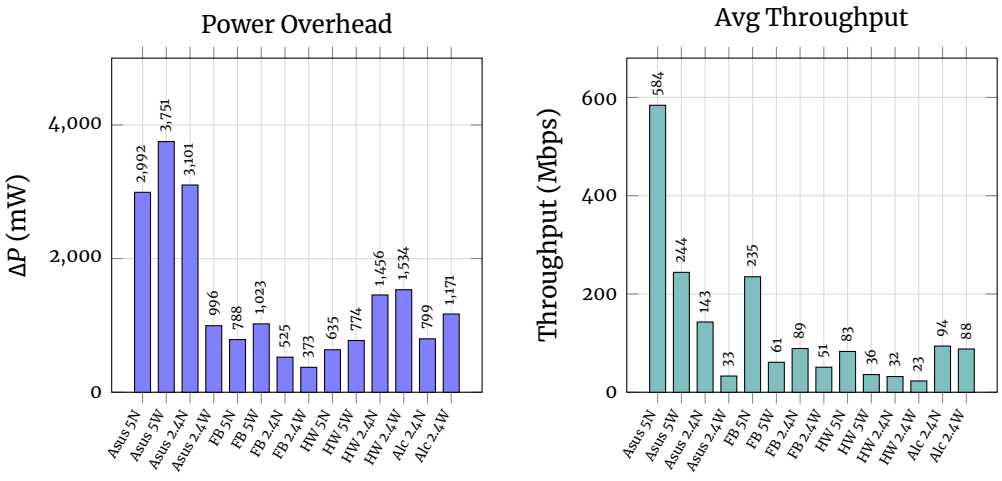
8.2 Power Overhead, Throughput, and Efficiency

Figure 12 compares the Wi-Fi power overhead (ΔP = load - idle), average throughput, and energy efficiency across all scenarios. N = near (<2 m), W = wall (~10 m, 1 wall); FB = Fritzbox, HW = Huawei, Alc = Alcatel.

Distance effect on power. There is no consistent rule here: moving the client behind a wall does not always increase router power. The Asus at 5 GHz is the clearest exception: power overhead *increases* from 2 992 to 3 751 mW (+25%) at the wall, even though throughput drops 58% (584→244 Mbps). The router is apparently boosting TX power to compensate for the worse signal. At 2.4 GHz the Asus does the opposite: overhead drops 68% (3 101→996 mW) because the link slows to 33 Mbps and the radio is mostly idle. The Alcatel draws 47% more at the wall with barely any throughput loss (94→88 Mbps).

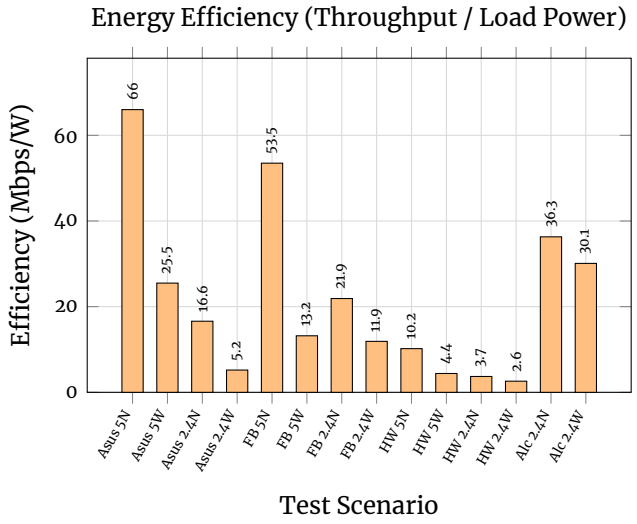
Band effect. At close range, 5 GHz is clearly faster: 2–4× higher throughput (Asus: 584 vs. 143 Mbps; Fritzbox: 235 vs. 89 Mbps), mainly thanks to wider channels. Through walls, 5 GHz degrades badly, as expected. The Fritzbox drops 74% on 5 GHz vs. 43% on 2.4 GHz.

Huawei variability. The Huawei results are all over the place: 0–253 Mbps swings within a single test run, 743–836 TCP retransmits. The phone-to-phone test topology is the culprit as both TX and RX use the wireless medium, effectively halving the available capacity and making the link more sensitive to interference.



(a) Wi-Fi power overhead (load - idle)

(b) Average Wi-Fi throughput



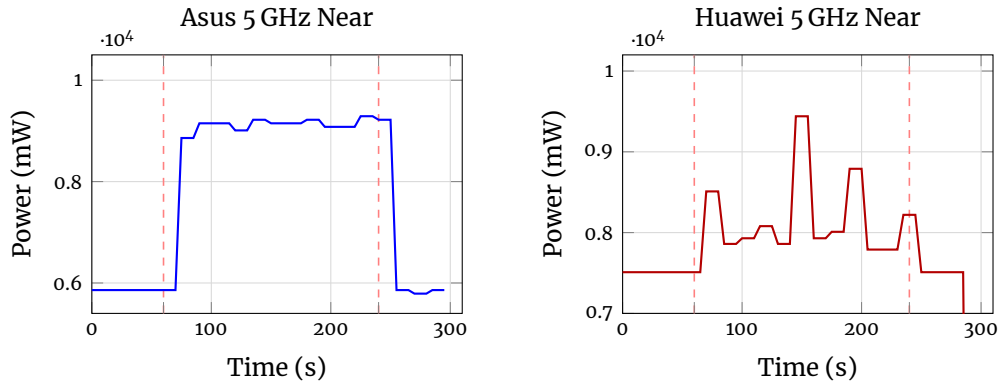
(c) Energy efficiency (Mbps/W of total load power)

Figure 12: Test 4 Wi-Fi comparison across devices, bands, and distances.

Time-Series Examples

8.3 RQ5: Comprehensive Wi-Fi Power Analysis

RQ5 summary. Wi-Fi costs 7–8× more power per megabit than wired Ethernet. The Asus draws 3.0–3.8 W for 244–584 Mbps on Wi-Fi, versus 444 mW for 1 Gbps wired. Distance effects are completely device- and band-dependent. 5 GHz is more efficient when you are close (Asus: 66 vs. 17 Mbps/W at near position) but degrades much faster through walls. The biggest single factor for throughput, and therefore efficiency, is channel width: the Asus runs 160 MHz on 5 GHz and hits 584 Mbps; the Fritzbox at 80 MHz reaches 235 Mbps; the Huawei manages only 83 Mbps, which is surprisingly low. Generation (ax vs. ac vs. n) matters less than how wide the channel is. And the absolute power spread is huge: at 2.4 GHz close range, the Alcatel draws 2.6 W total while the Huawei draws 8.7 W.



(a) Asus: 5.9 W idle → 8.9 W load (584 Mbps)

(b) Huawei: erratic load (WiFi-to-WiFi routing)

Figure 13: Representative WiFi time-series. Dashed lines mark iperf3 load start/end (60–240 s).

9. Results: ISP Bridge Mode Comparison

Test 5 answers a practical question: is it worth running an ISP modem in bridge mode alongside a dedicated router, from a power perspective? Configuration A is the Huawei operating standalone (reused results from Test 1). Configuration B is the Huawei in bridge mode with the Asus as the primary router. Both devices are plugged into one DECT 200 via a power strip, so the measurement captures combined system power.

9.1 Configuration B: Bridge Mode Results

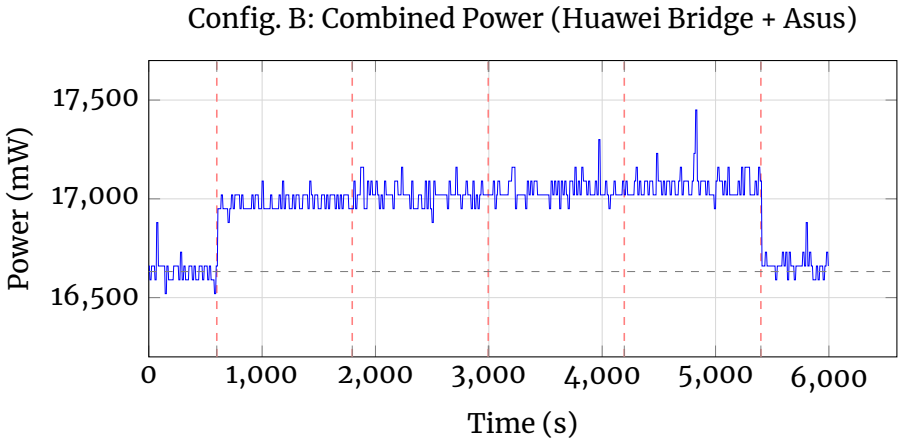


Figure 14: Test 5 Configuration B combined power.

Combined idle is 16 632 mW; full 4-port load reaches 17 072 mW (+440 mW, +2.6%). So the combined idle power is much higher then the Huawei alone (7 510 mW) and with the added load we see a much higher overall system power consumption than in Configuration A (17 072 vs. 7 490 mW). However, the incremental increase from load is only 440 mW, which is less than the Huawei

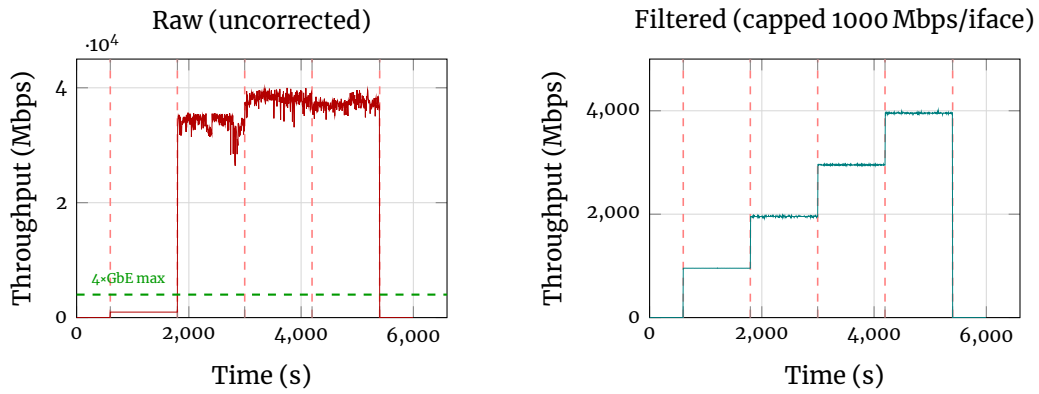
alone (1 534 mW). But overall a bridge mode setup is more power-hungry than a single device especially at idle and in low usage senarios.

Table 10: Test 5 Configuration B: Per-phase power and throughput statistics (combined Huawei bridge + Asus). Throughput values are filtered (capped at 1000 Mbps per interface; see Section 9.2).

Phase	Avg Power (mW)	σ (mW)	Min / Max (mW)	Avg Tput (Mbps)	Eff. (Mbps/W)
Pre-Test (Idle)	16 632	59	16 520 / 16 880	0	—
1 Port	16 981	51	16 660 / 17 090	955	56.2
2 Ports	17 021	58	16 880 / 17 160	1 951	114.6
3 Ports	17 045	57	16 950 / 17 300	2 951	173.1
4 Ports	17 072	70	16 950 / 17 450	3 951	231.4
Post-Test	16 673	77	16 590 / 17 090	0	—

9.2 Throughput Measurement Anomaly

The throughput data had a weird artifact: port 1 reported the expected ~955 Mbps, but the additional ports reported 12 000–33 000 Mbps each. What most likely happened is that the load generator was counting bytes written to the NIC's send buffer, not bytes that actually made it through the 1 Gbps WAN uplink. The NIC was just filling its DMA buffer as fast as it could (~35 Gbps aggregate = the Realtek RTL8125 ceiling shared across all active interfaces), completely ignoring the fact that the link couldn't actually carry that much. All per-interface values are capped at 1 000 Mbps for analysis, which worked surprisingly well as can be seen in Figure 15. **Power measurements were completely unaffected – the DECT 200 doesn't care what the software thinks the throughput is.**



(a) Raw: inflated by unsent-byte counting (b) Filtered: step-wise 955→3 951 Mbps

Figure 15: Test 5B throughput artifact.

9.3 Standalone vs. Bridge Mode Comparison

Table 11 puts the two configurations side by side. Huawei standalone (Config. A) vs. Huawei bridge + Asus (Config. B). The delta shows the additional power consumed by the two-device setup.

Table 11: Power comparison Configuration A vs Configuration B vs Double NAT.

Phase	Config. A Huawei (mW)	Config. B Bridge+Asus (mW)	Config. B vs. A Δ (mW)	Δ (%)	Double NAT* Huawei+Asus (mW)
Idle	8 962	16 632	+7 670	+85.6	17 188
1 Port	9 231	16 981	+7 750	+84.0	17 457
2 Ports	9 409	17 021	+7 612	+80.9	17 635
3 Ports	9 494	17 045	+7 551	+79.5	17 720
4 Ports	9 510	17 072	+7 562	+79.5	17 736
Post	8 986	16 673	+7 687	+85.5	17 212

*Calculated as Huawei Config. A + Asus standalone (Test 1) values combined.

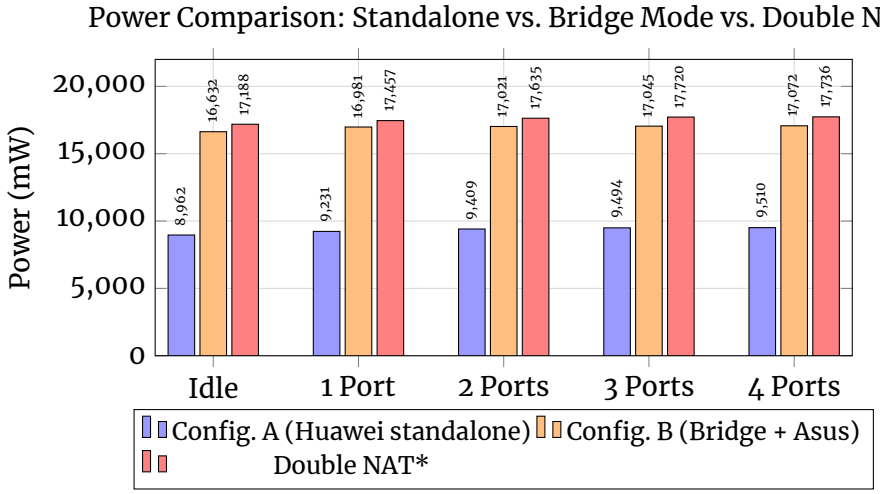


Figure 16: Standalone vs. bridge mode power vs. Double NAT. Config. B adds ~7.5–7.8 W.

9.3.1 Analysis (RQ8)

The two-device setup draws +7 670 mW (+85.6%) more at idle than the Huawei alone. Almost all of that is just the Asus existing: it draws 8 226 mW at idle regardless of what the Huawei is doing. The Huawei in bridge mode does save something compared to the data from Test 1. It draws ~556 mW less at idle and ~1 371 mW less at full load (6–14% reduction). But the Asus's 8.2 W idle is much higher than the savings.

RQ8 answer: It depends on the baseline interpretation. People usually use Bridge Mode because they *have* to use the ISP modem but *want* to use their own better router, giving them roughly three main options: 1. Huawei Standalone (Best power, least features, ~9 W). 2. Huawei Bridge + Asus Router (Moderate

power, all features, ~16.6 W). 3. Huawei Router + Asus Router chained in Double NAT (Worst power, double routing, ~17.2 W).

Bridge mode still saves power compared to running both devices in regular routing mode chained after each other (Double NAT). However, it is a significant energy loss compared to simply using the single unit provided by the ISP. The two-device setup costs about ~70 kWh extra per year compared to using only the Huawei modem. The main reason to do it is if one wants features the ISP modem doesn't provide, security features or performance features like better Wi-Fi coverage via a mesh network for example.

10. Conclusion and Outlook

10.1 Summary of Findings

To accurately answer the research questions regarding the power consumption of home to SME routers, a dedicated automated test framework was developed. Using a DECT 200 smart plug, a custom Go-based load generator, and a web-based analysis dashboard, five tests were run across four consumer-grade devices (`Fritzbox 7530`, Huawei EG8245W5-8T, `Asus RT-AX68U`, and `Alcatel HH40V`), addressing eight research questions.

Key findings:

RQ1: Port count has a modest effect – +6–16% above idle total, with each loaded port adding just 30–175 mW. The Fritzbox showed non-monotonic behaviour as its CPU saturated.

RQ2: An idle GbE cable costs 265–302 mW. For the Asus and Huawei, three idle cables cost *more* than running one port at full 1 Gbps load.

RQ3: Power scales sub-linearly – going from idle to line rate adds only 4.9–7.1%. Reducing idle power has a greater impact than limiting traffic.

RQ4: Bare-minimum power ranges from 1.14 W (Alcatel) to 6.34 W (Huawei); with all subsystems idle: 2.30–7.99 W, or €5–€18/year [7].

RQ5: Wi-Fi radios add 0.4–2.7 W idle and cost 7–8× more per megabit than wired Ethernet under load. Best efficiency: Asus 5 GHz near (66 Mbps/W); worst: Huawei (2.6–10 Mbps/W). Distance effects are device- and band-dependent.

RQ6: Fritzbox eco mode saves 474–670 mW (7–19%) but throttles Ethernet to ~100 Mbps/port. Asus and Huawei TX power reduction modes save negligible amounts considering their overall power consumption.

RQ7: Even at full combined load, no device exceeded 47% of its rated maximum. Spec sheet headroom is large.

RQ8: Bridge mode is not an energy-saving strategy compared to using a single ISP router. The Huawei + Asus combination draws ~16.6 W – +85.6% vs. the Huawei standalone at ~9.0 W. However, it still saves roughly 0.5 W compared to running the same two devices chained in a Double NAT configuration.

10.2 Future Work

- **Layer 2 switching tests**: Route traffic between two PCs *through* the DUT to test the switching fabric rather than the CPU.
- **Long-term idle runs**: A 24h+ measurement to check for sleep states or diurnal patterns.
- **More device categories**: Managed switches, mesh systems, enterprise APs, and dedicated unmanaged Ethernet switches to see if Wi-Fi replacement strategies scale similarly for dedicated switching hardware.
- **Additional load types**: ICMP, mixed TCP/UDP, real-world traffic captures.
- **ISP as Gateway tests**: More combinations of routers and bridge modes, to see if the Huawei + Asus result is typical or an outlier.

References

[1] Alcatel. 2018. Alcatel HH40V – User Manual. Accessed: 2025. (2018). https://www.alcatelmobile.com/lb/wp-content/uploads/2019/07/180514-HH40V-UM-DE-TR069.pdf.

[2] ASUSTeK Computer Inc. 2020. ASUS RT-AX68U – Technical Specifications. Accessed: 2025. (2020). https://www.asus.com/networking-iot-servers/wifi-routers/asus-wifi-routers/rt-ax68u/techspec/.

[3] AVM GmbH. 2018. AVM FRITZ!Box 7530 – Technical Data Sheet. Accessed: 2025. (2018). https://asset.conrad.com/media10/add/160267/c1/-/gl/804351123DS00/datasheet-1783471-avm-fritzbox-7530-wi-fi-modem-router-built-in-modem-adsl-vdsl-24-ghz-5-ghz-1200-mbits.pdf.

[4] AVM GmbH. 2024. AVM FRITZ!DECT 200 – Handbuch (User Manual). Accessed: 2025. (2024). https://cdn-reichelt.de/documents/datenblatt/E910/AVM_FRITZ_DECT_200_HB_DEU.pdf.

[5] Philipp Bornemann. 2023. go-fritzbox: Go library for Fritz!Box TR-064 and UPnP. Accessed: 2025. (2023). https://github.com/bpicode/fritzctl.

[6] ESnet / The Regents of the University of California. 2024. iPerf3 – The ultimate speed test tool for TCP, UDP and SCTP. Version 3.x. (2024). https://iperf.fr/.

[7] Eurostat. 2024. Electricity price statistics. Retrieved 03/24/2026 from https://ec.europa.eu/eurostat/statistics-explained/index.php/Electricity_price_statistics.

[8] Christian Feuersänger. 2022. PGFPlots: Create normal/logarithmic plots in two and three dimensions for LaTeX. Version 1.18. (2022). https://ctan.org/pkg/pgfplots.

[9] Home Assistant Community. 2024. Increase Fritz DECT polling intervall for power metering. Retrieved 03/24/2026 from https://community.home-assistant.io/t/increase-fritz-dect-polling-intervall-for-power-metering/679737.

[10] Huawei Technologies Co., Ltd. 2020. Huawei EchoLife EG8245W5-8T – Technical Specifications. Accessed: 2025. (2020). https://www.scribd.com/document/774390471/1593828050156465.

[11] 2015. LAN-Side DSL CPE Configuration. Technical report TR-064 Issue 2. Broadband Forum. https://www.broadband-forum.org/download/TR-064.pdf.

[12] uncle. 2024. iPerf – Network Speed Test (Android App). Accessed: 2025. (2024). https://play.google.com/store/apps/details?id=tools.uncle.network.