Managing Power Outages with a DIY UPS Monitor System

The Challenge: “Dumb” UPS Units in Small Networks

Many affordable uninterruptible power supplies (UPS) can keep equipment running for a while during an outage, but **they lack any communication interface** to signal connected devices about power loss. In a small office or home lab, you might have a UPS backing up critical devices (servers, network gear, etc.), but if that UPS **cannot tell your devices when it’s on battery and low**, those devices won’t shut down safely. This can lead to data loss or hardware damage when the battery finally dies. Buying a “smart” UPS with USB or network monitoring is expensive. What if we could add intelligent power management to a basic UPS at low cost?

A Two-Part Solution: UPS Server + UPS Clients

This DIY system solves the problem by using a clever two-part approach: a UPS Server and multiple UPS Clients. The UPS Server acts as a central brain that **detects a power outage** by monitoring simple clues in the network. The UPS Clients are lightweight scripts on each protected device that **listen for a signal** and shut down gracefully when necessary. Together, they provide a **low-cost, universal UPS monitoring solution** for any UPS, even without a data port.

How does it detect power loss? The UPS Server uses a “*canary in a coal mine*” strategy. Instead of talking to the UPS directly, it watches a set of always-on devices (called **“sentinel” hosts**) that run on normal mains power, not on the UPS. For example, these could be a couple of smart plugs, a modem, or any gadget plugged into the wall. As long as **at least one sentinel device responds** to network pings, the main grid power is considered on. But if **all the sentinels go dark simultaneously**, that’s a clear sign of a power outage. It’s like watching the streetlights go out – if everything on regular power stops responding, the UPS Server “knows” the electricity in the building has failed.

What happens next? Once a power loss is detected, the UPS Server broadcasts a status change to “**On Battery, Low Battery (OB LB)**.” This status is exposed through a standard UPS monitoring service (a virtual NUT server) that all UPS Clients are configured to check. Each UPS Client running on your critical devices will see the “OB LB” status from the server and then initiate its own shutdown countdown. For instance, a file server might start a 5-minute timer when it learns the power is out. If mains power returns in that window, **the UPS Server notices the sentinels come back** online and clears the alert, so the UPS Clients cancel their shutdown timers and keep running. But if the outage persists beyond the defined delay, each client safely shuts down its machine before the UPS battery is drained. This ensures critical systems are turned off cleanly, avoiding sudden power cuts.

Example Setup: Power Flow and Network Signals

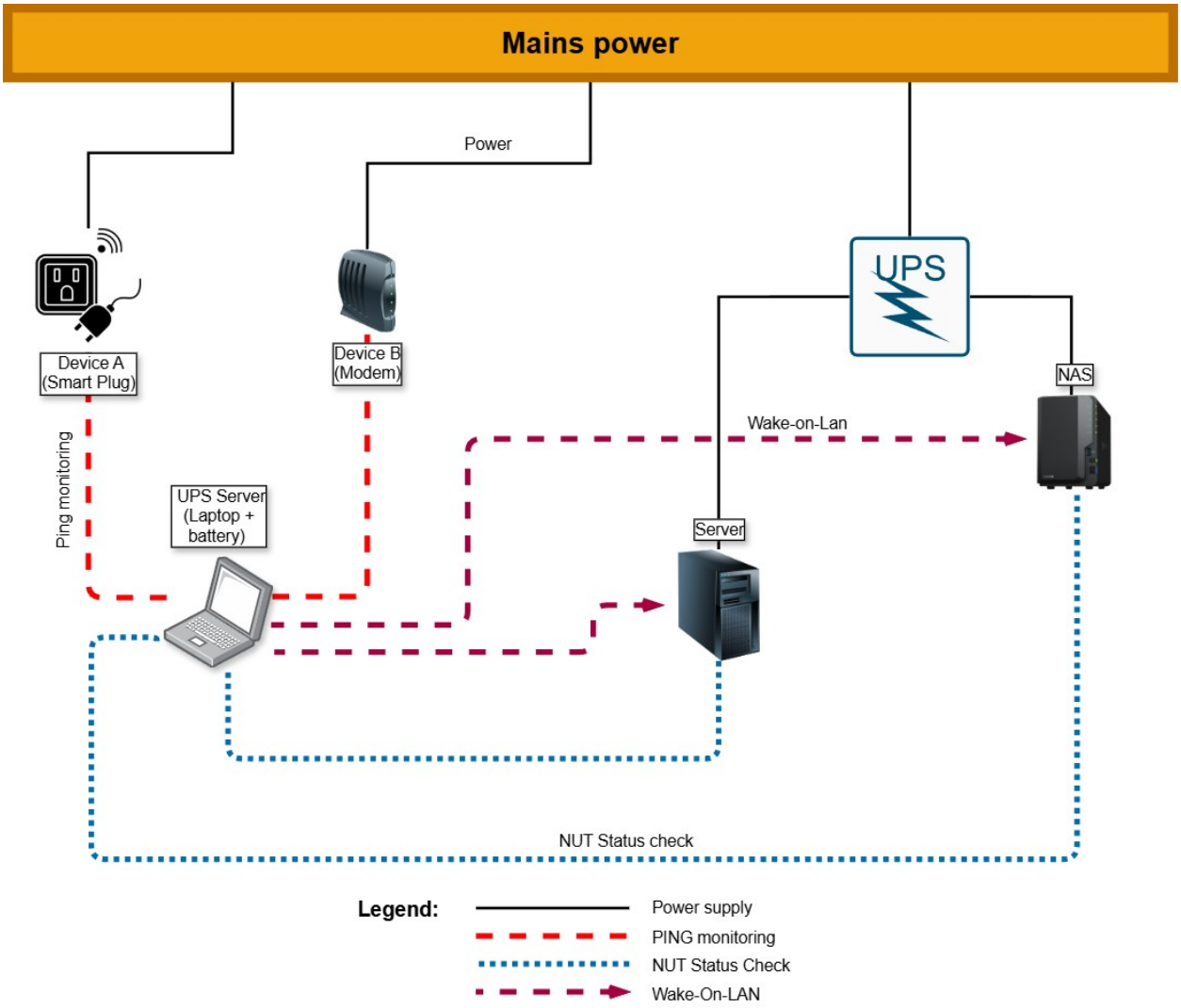

Figure: An example home network power setup. Solid lines (black) show electrical power connections; dashed lines (colored) show network communication. The UPS Server (e.g. a laptop with its own battery) monitors sentinel devices on the main power circuit. If all sentinels lose power, the UPS Server updates its virtual UPS status to "On Battery, Low Battery." UPS Client scripts on each critical device (servers, NAS) detect this status and after a preset delay, shut the devices down gracefully. Networking gear (router, switches, Wi-Fi AP) can be kept running on the UPS until the battery is exhausted. When mains power returns, the UPS Server switches the status back to "Online" and, after a short wait, sends Wake-on-LAN signals (purple dashes) to automatically reboot the shut-down devices.

In this diagram, the **mains electricity** feeds two parts of the network. First, it powers a couple of **sentinel devices** (say Device A and Device B) that are *not* on the UPS. Second, it feeds the **UPS unit**, which in turn supplies battery-backed power to the **critical devices** (like a server, a NAS, a router, etc.). The **UPS Server** (running on an independent power source, such as a laptop with its own battery or a Raspberry Pi on a separate supply) is connected

to the network. It continuously **pings the sentinel devices** (red dashed lines). Under normal conditions, at least one sentinel responds, indicating utility power is present. If the mains fails, those sentinels stop responding and the UPS Server immediately changes the UPS status to "On Battery, Low Battery," thinking: *"We're on backup power and the battery is running low"*. It doesn't actually measure the battery – it simply assumes that any outage should trigger an emergency shutdown as if the battery were low.

Meanwhile, each critical device has a UPS Client script that checks the UPS Server's status (blue dashed lines). They do this via standard Network UPS Tools (NUT) protocols or a simple API call, so it works with many platforms. Upon seeing the "OB LB" signal, clients don't shut down *immediately* – they start a **countdown as configured** (e.g. a few minutes). This allows short flickers or brief outages to pass without unnecessary reboots. If power comes back, the server's status returns to "Online" and the clients reset themselves, no harm done. If the outage persists, the timers run out and the clients execute a safe shutdown command, preserving data. Less critical machines power off first, conserving the UPS battery for truly essential gear (like networking equipment).

Throughout the outage, devices like the **router, firewall, or Wi-Fi access point** might be kept running on the UPS. They typically aren't running a UPS Client (no need to shut them down – you want internet/connectivity alive as long as possible). Those will ride out the battery until it's nearly depleted. Once the mains power is restored, the UPS Server detects the sentinels again and flips the status back to "Online" (meaning normal power). It then waits a bit (to ensure power is stable) and sends out **Wake-on-LAN packets** (purple dashed lines in the figure) to automatically boot up any machines that had shut down. In case some critical network gear did power off when the battery died, many devices like routers or switches can be configured to **auto-start when power returns** (either via BIOS settings or PoE behaviors), so the network comes back quickly. The result: your systems come back to life on their own, and you're back in business with minimal manual intervention.

Real-Life Use Case Example

Imagine a small office server room or a tech enthusiast's home lab. They have a **cost-effective UPS** (perhaps a DIY setup with a large battery inverter) that can hold up several devices for an hour, but it has **no USB or network port for monitoring**. On this UPS, they've plugged in a **Proxmox virtualization server**, a **NAS box**, a **router**, a **network switch**, and a **Wi-Fi access point**. Separately, on normal grid power, they have a couple of inexpensive **smart plugs** and an internet modem in another room – these will serve as the sentinels. They also repurpose an old **laptop (with its own battery)** to run the UPS Server software (perhaps via a Docker container).

One day, the building loses power. Instantly, the lights go out and the smart plugs go offline. The laptop running the UPS Server is unaffected (it's on its own battery); it pings the smart plugs and gets silence. It concludes the grid is down and marks the UPS status as on-battery. The Proxmox server and NAS, still running on UPS power, get the signal from the UPS Server that the power is out and battery is dwindling. Each begins a **timed shutdown sequence** as configured – the NAS might wait 2 minutes, the Proxmox server 5 minutes (to safely stop virtual machines). The router and switch, however, have no such shutdown script – they continue operating on UPS power to maintain network connectivity for as long as possible. After a few minutes, the NAS

and Proxmox cleanly power down, avoiding any data corruption. The router and Wi-Fi keep the network up until eventually, say 30 minutes later, the UPS battery finally depletes and they lose power too.

Now consider two scenarios: if mains power returns **before** the Proxmox server's 5-minute timer elapsed, the UPS Server would notice the sentinels back online and cancel the shutdown – the Proxmox and NAS would keep running continuously. But in this case, the outage lasted longer, so those systems did shut down. Later, when utility power is restored, the smart plugs come back online. The UPS Server detects this and switches status to "Online" (power restored). It waits a configured delay (maybe 5 minutes to be sure power is stable), then sends out Wake-on-LAN signals. The Proxmox server and NAS receive the "magic packet" and automatically boot up. Meanwhile, the router and switch, having been set to **auto-restart when power is back**, also come back on. In a few minutes, the whole network is up and running again without anyone having to press a power button. **This automated recovery is a huge benefit** – even if an outage happens while you're away, your systems shut down safely and come back on when power returns, all by themselves.

Key Benefits of this Approach

- **Works with Any UPS:** This solution is hardware-agnostic. It doesn't require a special USB or network-enabled UPS. **If your UPS can keep devices running, this system can manage them**, because it bases decisions on external power status, not a direct UPS data link. Even a DIY inverter or an older UPS model becomes "smart" with this setup.
- **Low Cost, High Flexibility:** You likely already have the ingredients – an extra Raspberry Pi or old laptop for the UPS Server and a few gadgets to act as power sentinels. The software is free (open-source) and containerized for easy deployment. There's no need to invest in enterprise-grade UPS systems for small-scale use cases.
- **Graceful Shutdown = Data Safety:** By performing controlled shutdowns, the system prevents data corruption and hardware stress. For devices like NAS units or hypervisor hosts, this is critical. The method avoids the pitfalls of some built-in UPS clients that often hang or fail in custom setups. Instead, it uses simple, reliable commands (like a direct `shutdown` call) after a countdown, and it automatically cancels if power returns in time. This **robust and predictable behavior** has been tested on various platforms (Synology DSM, Proxmox VE, standard Linux, etc.).
- **Automatic Recovery:** Beyond shutdowns, the inclusion of Wake-on-LAN means your systems can come back online without manual intervention. This is especially useful for remote or unmanned sites – you don't have to be physically present to power things back on. Your critical network infrastructure stays up as long as possible and recovers as soon as it can.
- **Centralized Management:** The UPS Server acts as a single hub that not only detects outages but also serves a **central API for client configuration**. In practice, this means you can manage all your client shutdown timers and settings in one place (for example, via a simple config file or REST API on the server). The clients periodically fetch their configuration, so any adjustments (like changing a shutdown delay) can be done centrally. This eases the administration of multiple devices.

- **Easy Deployment:** The server component is packaged as a Docker container for convenience. You can run it on any machine that's on the UPS (or a separate battery-powered device). The clients are lightweight scripts (a few kilobytes) that can be installed on Linux boxes or even on NAS operating systems with minimal hassle. There's no heavy agent or complex installation – just a cron job or scheduled task running the script every minute. This simplicity means fewer points of failure.

Who Can Benefit?

This solution is ideal for **home labs, small businesses, or DIY enthusiasts** who use a UPS but don't have high-end models with built-in monitoring. If you have multiple computers, servers, or IoT devices running through frequent outages or unreliable power, implementing this system can save you from headaches. For example:

- A **home automation setup** with a modest UPS: You might have a home server and some smart home hubs on a UPS while the rest of the house isn't. Using this system, your server can shut down nicely during a blackout, and your critical hubs stay on as long as possible.
- A **small office with a network closet**: The office has a couple of critical PCs and a network switch on a UPS, but the UPS can't signal. Deploy a UPS Server on an always-on micro PC. Let some wall-powered devices (like a printer or clock) be sentinels. In an outage, office PCs save their work and turn off properly, while networking gear continues until power is nearly gone – keeping the office Wi-Fi and phones alive for as long as the UPS holds.
- **Remote installations**: Think of a remote telecom tower or an edge server cabinet with a basic UPS. You could use a cellular router (on UPS) and a remote IP power monitor (on mains) as the sentinel. The UPS Server could run on a tiny single-board computer. When the grid fails, the edge servers shut down safely. When power returns, everything powers back up, **avoiding a site visit to reboot equipment**.

In all these scenarios, the **DIY UPS monitor system adds intelligence to an otherwise "blind" UPS**. It's a **practical, low-budget enhancement** that can make your power backup setup far more robust and autonomous.

Conclusion

Power outages are unpredictable, but with a smart monitoring approach, even a simple UPS can become part of an automated disaster recovery plan. This two-module system (UPS Server + UPS Clients) provides peace of mind by ensuring your devices know when to shut down and when to power back up, all without expensive hardware. By using network pings as a proxy for power status and leveraging standard tools for shutdown and reboot, the solution remains elegant and vendor-neutral. Whether you're guarding a professional server rack or just a home media center, this concept can be a lifesaver – preventing data loss, extending battery life for what's truly important, and getting your world back online with minimal fuss once the lights come back on. It's a great example of creative problem-solving in the tech world: using software and clever architecture to fill the gaps of hardware limitations. **In short, it keeps your systems**

running when everything else is dark, and gets them back on when the power returns – all automatically.

For more details see:

https://github.com/MarekWo/UPS_Server_Docker
https://github.com/MarekWo/UPS_monitor