What Keeps Your Network up at Night?

Author(s):
Röllin, Lukas; Jacob, Romain; Vanbever, Laurent

Publication date:
2023-12-05

Permanent link:
https://doi.org/10.3929/ethz-b-000649098

Originally published in:
https://doi.org/10.1145/3624354.3630092

Poster: What Keeps Your Network up at Night?

Lukas Röllin
ETH Zürich
Switzerland

Romain Jacob
ETH Zürich
Switzerland

Laurent Vanbever
ETH Zürich
Switzerland

ABSTRACT

The demand for ever-faster links and devices is competing with the need to make networks more energy-efficient. One energy-saving technique is shutting down dispensable parts of the network. In this work, we investigate the limits of link sleeping; *i.e.*, turning off underutilized links. We show that turning transceivers on takes on the order of seconds, making them the bottleneck for fast network wake-up. We also present a *power plane* prototype designed to orchestrate link sleeping alongside standard routing protocols.

CCS CONCEPTS

• **Networks** → **Network protocols**.

ACM Reference Format:
Lukas Röllin, Romain Jacob, and Laurent Vanbever. 2023. Poster: What Keeps Your Network up at Night?. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies (CoNEXT Companion '23), December 5–8, 2023, Paris, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3624354.3630092

1 INTRODUCTION

Previous research on making networks more energy efficient falls into two categories: making the individual parts more efficient or turning parts that are not needed off. In wired networks, turning off links is promising for two reasons. One is the observation that many network links are underutilized (<30%) because network operators typically plan for peak demand. The other reason is that transceivers waste a lot of energy when idle. For example, we measure 5W idle power vs. 6.2W under full load for a 100G LR4 optical transceiver. In other words, 80% of its total power is spent for no useful work.

Previous research on link sleeping made two assumptions that do not hold in today's networks. One is that interfaces can wake up within a few milliseconds; in fact, we observe they take three orders of magnitude longer (Fig. 1). This makes many previous ideas, *e.g.*, waking up for each incoming packet or buffer and burst traffic [1], infeasible. Another unrealistic assumption is that the required network state information is instantaneously available to the controller. Some works even assume reliable predictions of future network demands, which is very challenging—at best.

Our work aims to identify the limits of today's networks and design a realistic link sleeping system. In summary:

- We present a first prototype of a *power plane* responsible for optimizing the network energy usage, which is seamlessly integrated between the standard data and control planes (§ 2).
- We measure interface wake-up delays (§ 3) and show that milliseconds wake-up times remain impossible today.
- Nonetheless, we show it is possible to implement link sleeping in hardware if one considers larger time scales (§ 3).

This work helps estimate how often we can set links to sleep and how long it takes for the network to react to traffic changes. Sadly, we cannot translate sleeping times into energy savings—yet—as this requires power models that are currently unavailable. Establishing such models is part of parallel research efforts.

CoNEXT Companion '23, December 5–8, 2023, Paris, France

ACM ISBN 979-8-4007-0407-9/23/12.
https://doi.org/10.1145/3624354.3630092

2 POWER PLANE DESIGN

The power plane's overall mission is to put as many links to sleep as possible with minimal disruption to the network. In other words, the power plane has three objectives.

(1) It must *not disconnect the network* by putting links to sleep.
(2) It must *decide which links to put to sleep* while minimizing traffic redirection and congestion.
(3) If congestion happens, it must *react quickly* and turn links back on to resolve the congestion.

While we aim to set as many links to sleep and keep them asleep for as long as possible, avoiding congestion takes priority.

We consider a network running OSPF with a centralized power plane controller deciding which links to put to sleep and a decentralized wake-up mechanism allowing any node to initiate a network-wide wake-up. We use OSPF because it is a widely used intra-domain routing protocol that is capable of distributing network link utilization through the OSPF TE Metric Extensions (RFC 7471); this gives the power plane a complete view of the link loads within the network, albeit with some delay.

A centralized controller easily avoids accidental network partitions and synchronizes the two sides of a link; the controller always knows which links are asleep and which must stay up to keep the network connected. Guaranteeing connectivity is non-trivial with a decentralized sleeping mechanism. Listing 1 summarizes the controller logic, for more details see [2].

The network wake-up is decentralized to speed up the reaction to congestion. Instead of waiting for OSPF to distribute the link load information in the network, each router can start a wake-up event independently. If a router detects high utilization or a link failure on its interfaces, it starts broadcasting wake-up commands within the

Listing 1: Sleep Decision Pseudocode

```
# 1. Check link utilization and available bandwidth to reroute
for link in read_ospf_database().links:
    # Minimize rerouting: do not sleep links with utilization higher
    # than the utilization threshold parameter max_util(0.4)
    if link.utilization > max_util: continue
    # Avoid congestion: only sleep if enough capacity + margin(0.2)
    minimum_available = check_reroute_capacity(link)
    if minimum_available > link.utilization + safety_margin:
        mark_link_to_sleep(link)
# 2. Sort link by utilization and avoid disconnected network
sleep_links = check_connected(sort_by_util(links_marked_to_sleep))
```

network. The centralized sleeping controller also receives the wake-up message such that it does not try to turn off links simultaneously. The sleep and wake commands are sent to a script that turns links on and off via the router CLI.

3 PRELIMINARY EVALUATION

Interface Wake-up Time. Previous work assumes interface wake-up times of a few milliseconds, but our measurements show that they take on the order of seconds. We measure different types of interfaces and find that 100G optical transceivers take roughly 8–15 seconds to wake up, while electrical ones are a bit faster, taking around 2–8s (Fig. 1). We also confirm that the wake-up delay is independent of the sleep time; *i.e.*, even if sleeping for just 50ms, it takes seconds to wake up the transceiver (not plotted).

In summary, one should be careful when turning links off, as reverting the decision takes a long time.

Network Wake-up Bottlenecks. To test our power plane design without access to a complete network, we use a tool called the mini-internet. This tool emulates a network (8 nodes) on one machine using docker containers and virtual interfaces. Compared to simulators, the mini-internet runs standard Linux network stacks and emulates link delays and buffers, resulting in realistic evaluations.

Our experiments show that the interface wake-up delay is by far the slowest part of waking up the network in the case of congestion (Table 1). The four steps of waking up are detecting the congestion, flooding the wake-up commands within the network, waking up the physical interfaces and reconverging the routing state. Detecting congestion takes roughly 1s and is mainly limited by the frequency at which the router updates its link load value. While increasing the update frequency could make the reaction time faster, it increases the risk of false positives due to bursty traffic. Flooding the wake-up commands within the network is limited by how long it takes for a packet to reach the furthest node—at most a few 100ms in most networks. The most significant bottleneck with current hardware is the wake-up delay of the interface: several seconds (Fig. 1). The last step to use the awakened link is for OSPF to distribute the new link state, calculate the shortest path, and update the data plane.

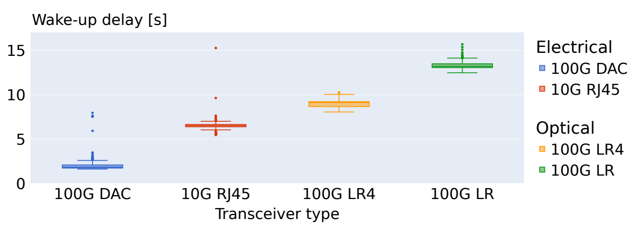

Figure 1: The wake-up time is on the order of seconds, not a few milliseconds, as assumed in previous research.

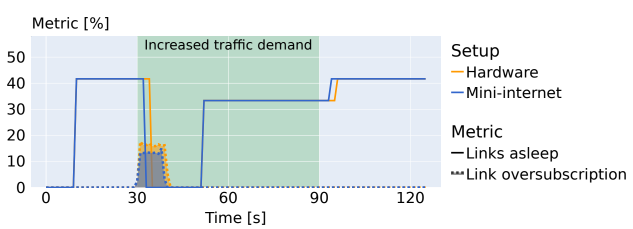

Figure 2: Our power plane prototype operates as expected. The behavior on the hardware matches the emulation well.

Table 1: Our measurements show that the link wake-up delay is the slowest part of the wake-up process.

Steps	Scale	Limitation
Detection	~1s	frequency of load measurement
Flooding	~0.1s	network diameter
Wake-up	~1–10s	interface wake up delay (Fig. 1)
Convergence	~0.1s	OSPF parameters & computation

When using default OSPF parameters, the convergence takes about 10s but one can cut it down to a few 100 ms by tuning some parameters. We achieve this by decreasing the `hello` interval to 0.1s, the `dead` interval to 1s and setting the link type to point-to-point. We also lower the shortest path calculation and LSA throttling timers to a few 10s of milliseconds. Changing those parameters increases the network usage of OSPF, but our measurements show that each link's average OSPF traffic reaches 32 kbit/s, which is negligible compared to today's network bandwidths.

Hardware Validation. We implement our controller on hardware using two Cisco Nexus 9300 routers to validate the results from the mini-internet setup. We use VRFs to create eight virtual nodes out of our two routers. As these routers do not support the OSPF TE metrics extension, we manually create the TE metrics LSAs but still rely on OSPF to distribute them in the network as per the RFC.

The scenario contains a base traffic demand that allows the controller to put links to sleep at first (Fig. 2). After 30s, one flow increases for 60 seconds; it creates congestion, and the power plane reacts by waking up the whole network. Then, at 50s the controller puts some links back to sleep under the new traffic demand. Since the traffic demand is higher now, it can put fewer links to sleep than before. At 90s, the traffic demand decreases and the controller puts more links back to sleep (Fig. 2). Our power plane controller behaves similarly in our hardware and emulated setups. This suggests that (i) we can design power plane controllers that are compatible with today's hardware and standard protocols; (ii) we can hope emulation remains realistic for experiments with more complex topologies and traffic patterns.

4 CONCLUSION AND FUTURE RESEARCH

Our first power plane prototype shows that link sleeping is possible, but it must be considered at timescales of tens of seconds and above.

Future work should explore the power plane design space, including the potential benefits of enhanced network state visibility, which could enable better sleeping decisions.

Another important direction is quantifying the energy savings and network disturbance resulting from link sleeping. The two missing pieces are detailed traffic traces and network power models, which we both try to acquire in our ongoing research.

REFERENCES

[1] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. 2008. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*.

[2] Lukas Röllin. 2023. *How Fast Can It Wake Up? Putting Link Sleeping into Practice*. Master's thesis. ETH Zürich. https://doi.org/10.3929/ethz-b-000637984