# SynchroNB: Toward Robust Timing for 5G NB-IoT Networks

Muhammad Abdullah Soomro*
University of Massachusetts, Amherst
msoomro@umass.edu

Muhammad Shayan Nazeer*
University of Massachusetts, Amherst
mnazeer@umass.edu

Collin DelSignore
University of Massachusetts, Amherst
cdelsignore@umass.edu

Yasra Chandio
University of Massachusetts, Amherst
ychandio@umass.edu

Muhammad Taqi Raza
University of Massachusetts, Amherst
taqi@umass.edu

Fatima Muhammad Anwar
University of Massachusetts, Amherst
fanwar@umass.edu

## Abstract

Emerging resource-constrained cellular Internet of Things (IoT) applications such as drone swarms, autonomous vehicles, and remote surgery via mixed reality demand millisecond-level time synchronization. Narrow-Band IoT (NB-IoT), the leading low-power wide-area technology, struggles to meet these requirements. The root cause lies in the non-deterministic delays inherent in the 5G protocol design. Uplink reliability and scheduling mechanisms introduce asymmetric latencies that disrupt conventional time synchronization algorithms such as the Network Time Protocol (NTP). Time-critical packets are further affected by deep-sleep wake-up latency, base station scheduling delays, uplink/downlink asymmetry, and unpredictable drift from inexpensive oscillators. Together, these factors can accumulate into timing errors on the order of hundreds of milliseconds. In this paper, we first quantify timing errors across five dimensions on a commercial NB-IoT network. We then present SynchroNB, an on-device framework that combines lightweight machine learning with a cross-layer control loop. SynchroNB forecasts 5G network volatility and crystal drift to adaptively wake the cellular modem, reserves uplink resources just in time, switches into resilience mode when the wireless link degrades, and prioritizes time synchronization packets in the MAC-layer queue. We deploy SynchroNB on commercial NB-IoT hardware and evaluate it over a live 5G network. Our experiments show that SynchroNB achieves single-millisecond-level synchronization accuracy under *NB-IoT uplink/downlink asymmetry* and, diverse wireless conditions, while requiring only 36% of the radio-on time and 25% of the bandwidth of the NTP baseline, transforming NB-IoT time synchronization from a reactive protocol into an intelligent, self-tuning control loop.

## Keywords

5G, NB-IoT, Cellular IoT, Time Synchronization, Machine Learning

**ACM Reference Format:**
Muhammad Abdullah Soomro, Muhammad Shayan Nazeer, Collin DelSignore, Yasra Chandio, Muhammad Taqi Raza, and Fatima Muhammad Anwar.

*Both authors contributed equally to this research.


*Conference'17, Washington, DC, USA*

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

**Table 1: Time-Critical Applications in Cellular NB-IoT**

| Application | Sync Req. | Impact of Misalignment |
|---|---|---|
| Smart Grid | <20 ms | Delayed fault isolation, equipment damage. |
| Mixed Reality | <20 ms | Nausea, disorientation, poor UX. |
| Industrial Autom. | 1–10 ms | Robotic faults, safety hazards. |
| Drone Coordination | <10 ms | Collisions, unstable formations. |
| 5G Positioning | 1–10 ms | Location errors, unsafe guidance. |

2026. SynchroNB: Toward Robust Timing for 5G NB-IoT Networks. In . ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Modern cellular Internet of Things (IoT) platforms are no longer confined to low-rate telemetry; they are rapidly extending into critical and time-sensitive domains such as mixed reality [22], precision drone swarms [27], and other applications detailed in Table 1. These applications depend on precisely coordinated actions, distributed sensing, multi-sensor fusion, event ordering, synchronized sleep-wake cycles, and secure handshakes in cryptographic operations [16, 35, 45, 56], that demand end-to-end time synchronization on the order of a few milliseconds [7, 25]. Even a handful of milliseconds of misalignment can lead to anything from unstable swarm flights [55] to user motion sickness [31, 48], yet they are typically deployed in remote or battery-powered environments where wired or high-throughput connectivity is infeasible. Narrowband-IoT (NB-IoT) [64] offers deep-coverage, wide-area scalability, deep sleep modes [33, 47] and ultra-low transmit power that together can stretch battery life to five years or more on a single cell [53], making it a viable choice for such deployments. In essence, this work focuses on applications that benefit from NB-IoT's extended battery life and wide coverage, rather than those requiring high throughput or low latency, which are better supported by 5G NR or RedCap. These constrained deployments require reliable timing without network modification.

However, maintaining accurate clocks across these battery powered deployments remains challenging [5, 30]. Today's NB-IoT devices typically rely on the Network Time Protocol (NTP) [40] or its simpler variant SNTP [41] to periodically poll external servers for time. In theory, NTP can maintain millisecond-level sync on wired or Wi-Fi networks [39]. In practice, however, we find that ***directly applying NTP over NB-IoT yields time offsets more than 100 times worse***. Fundamentally, an NB-IoT device's clock can fail to sync for two main reasons: (1) *highly asymmetric wireless link delays* where the one-way latency for sending a timestamp (uplink) can vastly exceed that of receiving one (downlink) [54], and (2) *inexpensive and inaccurate oscillators* that slowly gains or loses time [63]. NTP misinterprets extra uplink latency as clock

drift and (*attempts to*) correct the clock in the wrong direction [15], or refuses to adjust when it should [44], leading to synchronization errors far outside acceptable error bounds. *Aggressive power-saving modes* further aggravate the problem: each wake-up from deep sleep (i.e., the idle-to-connected transition) incurs a multi-hundred-millisecond handshake, which NTP misinterprets as clock skew. This results in large, unnecessary time corrections on every resume procedure [54]. The lack of robust fail-safe mechanisms in conventional network-layer designs, such as static timeouts or conservative safety margins, makes it difficult to correct these errors [37].

In our empirical analysis, we show that existing approaches to time synchronization in cellular IoT fall short under NB-IoT's constraints (§4). The off-the-shelf NB-IoT module using SNTP saw its clock out of sync by 100s of millseconds during various network conditions because NTP [40] assumes symmetric delays and constant radio availability, misestimate when devices sleep for minutes at a time, or face narrowband retransmissions [44]. On the other hand, high-accuracy 5G solutions such as TSN/IEEE 802.1AS deliver sub-microsecond accuracy [50], but they demand hardware timestamping, dedicated slices, and low-latency links that NB-IoT cannot support [60].

Existing wireless sensor network (WSN) protocols such as Zigbee [12] and BLE mesh networks [6] use peer-to-peer beaconing [11, 18] or multi-hop consensus [59] for timing. These approaches assume devices can listen to each other and exchange timestamps. In NB-IoT, devices cannot do this as the network uses a strict star topology controlled by the base station, devices sleep for long periods, and cannot timestamp each other's traffic [4]. As a result, broadcast and peer-based synchronization methods do not apply. LoRa, while also operating in a star topology, depends on network infrastructure for synchronization, e.g., LoRaWAN gateways periodically broadcast timing beacons that end devices use to align their clocks [46]. Such gateway-assisted methods are unavailable in NB-IoT. These constraints motivate a *device-only* approach that predicts timing asymmetry locally without infrastructure hooks.

In this paper, we present SynchroNB as the first device-only synchronization framework for NB-IoT that explicitly models uplink/downlink asymmetry without requiring network or protocol modifications. Instead of treating NTP as a black box and hoping more frequent polls or network upgrades will help, we tackle the problem holistically from the device side. Our solution is *an on-device framework that proactively anticipates and compensates for both clock drift and network asymmetry* ***before*** *they impact sync accuracy.* Our design blends lightweight machine learning with cross-layer control in the NB-IoT modem, all within the energy and bandwidth constraints of IoT devices.

At its core, SynchroNB orchestrates two lightweight on-device predictors in tandem. The *drift analyzer* continuously learns each device's crystal characteristics, forecasting when its clock will exceed a specified error threshold. Simultaneously, the *network volatility predictor* monitors radio link quality to estimate when a single-packet SNTP exchange may be distorted by transient channel conditions. In response to these predictions, SynchroNB employs an adaptive wake-up scheduling policy that proactively activates the modem and injects an uplink beacon, mitigating cold-start latency. It then pre-allocates packet resources through a closed-loop reservation scheme, ensuring immediate uplink grant availability and further reducing delay asymmetry. Grant acts as a permission from the base station that allows a device to transmit data on the uplink. When link conditions deteriorate, SynchroNB dynamically switches to a resilient burst synchronization mode, transmitting multiple time packets in rapid succession and filtering for the most reliable round-trip times. Finally, it introduces a priority assignment mechanism within the modem's MAC queue to prevent head-of-line blocking, guaranteeing timely delivery of time packets while preserving fairness across other traffic classes.

These mechanisms run entirely on the device's microcontroller unit (MCU) and modem firmware, requiring minimal extra bytes, a few microseconds, and occasional packets, with no changes to the 5G network infrastructure. In essence, our approach transforms the NB-IoT time sync problem from a passive, reactive task (periodically asking an NTP server and hoping for the best) into an intelligent, device-driven control loop that actively mitigates the known timing-related failure modes of the underlying network. To this end, we make the following contributions:

(1) **Problem characterization of NB-IoT time sync (§3):** We present an in-depth theoretical and empirical analysis of why conventional NTP fails on NB-IoT, quantifying the impact of oscillator drift and multi-fold form of network asymmetry on synchronization error. Through empirical measurements on a commercial NB-IoT network operator, we show that naive SNTP can suffer timing errors up to several seconds. We distill five fundamental challenges that a practical and reliable solution must overcome.

(2) **SynchroNB On-device synchronization framework (§5):** We design and implement a lightweight synchronization framework that mitigates NB-IoT uplink/downlink asymmetry through adaptive, device-side control. Our solution integrates lightweight machine-learning predictors with a suite of cross-layer optimizations. Unlike prior methods, it requires no hardware timestamping or network support beyond standard NB-IoT connectivity. The device itself learns its clock drift (using a compact LSTM-based model) and detects link anomalies (with a temporal convolutional network) to dynamically adjust its synchronization strategy.

(3) **System implementation (§6) and evaluation (§7):** We develop a full prototype on commercial NB-IoT hardware and a lightweight cloud NTP server. We evaluate the system in diverse scenarios, varying signal conditions, network loads, and application traffic patterns and demonstrate an order-of-magnitude improvement in sync accuracy compared to standard NTP. We show that our solution's benefits come with minimal overhead. Our adaptive algorithms limit extra transmissions resulting in negligible additional bandwidth use.

## 2 Related Work

NB-IoT's time synchronization challenge differs fundamentally from both *mesh-based* wireless sensor networks that rely on cooperative peer exchanges and *infrastructure-assisted* star-topology systems such as LoRa or industrial TSN, where the base station or coordinator provides timing beacons. In this section, we review

existing synchronization methods across these categories and highlight why they cannot be directly applied to NB-IoT's constrained, device-centric architecture.

NB-IoT time synchronization relies on general methods such as Network Identity and Time Zone (NITZ) [1] and the Network Time Protocol (NTP) [40], while advanced industrial cases may use Time-Sensitive Networking (TSN) [60]. NITZ offers coarse, infrequent updates tied to signaling events, unsuitable for continuous alignment. NTP provides millisecond accuracy via periodic timestamp exchanges over IP but we have shown in this paper that it struggles with NB-IoT's deep sleep and uplink constraints, reducing accuracy. TSN delivers precise timing for 5G URLLC but requires tightly coordinated scheduling and high bandwidth, making it incompatible with NB-IoT. Thus, NITZ is specific but coarse, NTP is general but unreliable in NB-IoT, and TSN is precise but incompatible. Figure 1 shows timing solutions for NB-IoT across the accuracy-complexity spectrum and puts our proposed solution in context to state-of-the-art NB-IoT options.

Time synchronization in other IoT radio technologies such as IEEE 802.15.4-based Zigbee, and BLE Mesh often leverages periodic beaconing, time-stamped messages, or distributed consensus protocols [11, 18, 59]. For instance, WSNs traditionally employ protocols such as Reference Broadcast Synchronization (RBS), Timing-sync Protocol for Sensor Networks (TPSN), or Flooding Time Synchronization Protocol (FTSP) to achieve sub-millisecond accuracy under favorable link conditions [19, 21, 38]. These approaches assume peer-to-peer communication and frequent coordination, both impractical for NB-IoT devices operating under deep-sleep and scheduling constraints.

In contrast, long-range IoT systems such as LoRaWAN also follow a star topology. However, unlike NB-IoT, LoRa networks provide infrastructure support for synchronization. Schemes such as LongShoT and TS-LoRa use gateway broadcasts or PHY-level slot timing to align device clocks [46, 65]. NB-IoT devices, although similarly organized in a star topology, do not have access to such timing beacons or network-level broadcasts, and cannot observe or correct timing at the PHY layer. This lack of infrastructure assistance further motivates our *device-only* synchronization approach.

Crystal clock drift prediction and compensation have been studied extensively in prior work. Drift has two parts: an approximately linear offset and a random jitter. It is typically modeled using Kalman filters [26, 29] or regression models [51, 61]. These methods work well for predicting the slow, long-term component of drift, but they struggle with short-term changes caused by jitter. As a result, to stay accurate, they usually require frequent timestamp updates and continuous observations. NB-IoT devices, however, spend long periods in DRX/PSM sleep and cannot sync at fixed intervals without paying a high energy cost. There are also external compensation methods, such as using on-board temperature sensors to correct the oscillator [52] or having the network gateway estimate drift for nodes [58]. These approaches rely on extra sensors or infrastructure support. Our constraints differ: we require a method that runs on the device alone and does not assume any changes to the NB-IoT network. These constraints motivate us to adapt the drift-prediction mechanism and build a refined model tailored to NB-IoT.

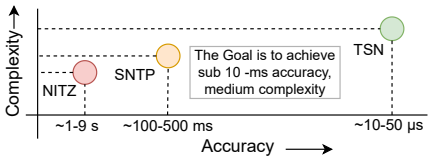



**Figure 1: Cellular IoT time synchronization options**

Existing solutions for time synchronization either depend on peer coordination (WSN/mesh), gateway beacons (LoRa/ TSNs), or frequent timestamp updates (Kalman/regression models). NB-IoT devices cannot rely on any of these, as they cannot overhear peers, they do not receive dedicated timing beacons from the network, and they cannot afford constant resynchronization. Our work, therefore, targets a different point in this space: device-only synchronization for NB-IoT that operates under long sleep intervals, asymmetric uplink/downlink timing, and without infrastructure modification.

## 3 Timing in Cellular NB-IoT

Timing packets (from NTP or SNTP) experience delays within NB-IoT protocol layers before transmission and after reception. We will show that the one-way delays are highly asymmetric because NB-IoT uses different processing paths for uplink and downlink.

**Uplink Processing Delays.** On an uplink path when an IoT device is in Power Saving Mode (PSM), the 5G network uses Discontinuous Reception (DRX) cycle and wakes only at predetermined intervals to check for, or initiate, traffic [33]. The time packet generated in the application layer ① waits idle during the DRX ramp-up (refer Figure 2) interval, $T_{DRX}$ ②, while the modem powers up the RF front-end and transitions from the `RRC Idle` to `RRC Connected` state. Even with established radio connections, no uplink (UL) resources have yet been assigned. Before transmission can begin, the device must obtain permission from the network base station to access the wireless channel. It waits until the next UL opportunity in the TDD [1] frame, an additional delay of $T_{SR}$ ③ and then transmits a *Scheduling Request* (SR) ④. On receiving SR, the base station allocates a specific time-frequency resource block and returns the corresponding *UL grant* ⑤. The device gets a UL grant after $T_{grant}$ ⑥. After obtaining valid grant, device moves its packet from the MAC buffer to the PHY, and modulates the transport block to transmit over the air on the allocated radio resource ⑨.

The MAC layer schedules the packets for transmission over the air in the MAC buffer. Through various logical channels, MAC distinguishes between control and data traffic and schedules using *Logical Channel Prioritization Scheme*. In case multiple applications are sending data over the network, the time packets sometimes have to wait at MAC buffer to be scheduled for transmission ⑦ with $T_{MACsched}$ latency. If the device-transmitted signal corrupts over the wireless channel and the base station cannot decode it, it requests a retransmission through a process called Hybrid Automatic Repeat Request (HARQ). Each retransmission incurs additional latency of $T_{HARQ}$. Therefore, the total UL latency is:

$$T_{Uplink} = T_{DRX} + T_{SR} + T_{grant} + T_{MACsched} + no.Retx \times T_{HARQ}$$

[1] 5G uses Time Division Duplex (TDD) to allocate uplink (UL) and downlink (DL) slots for network users.

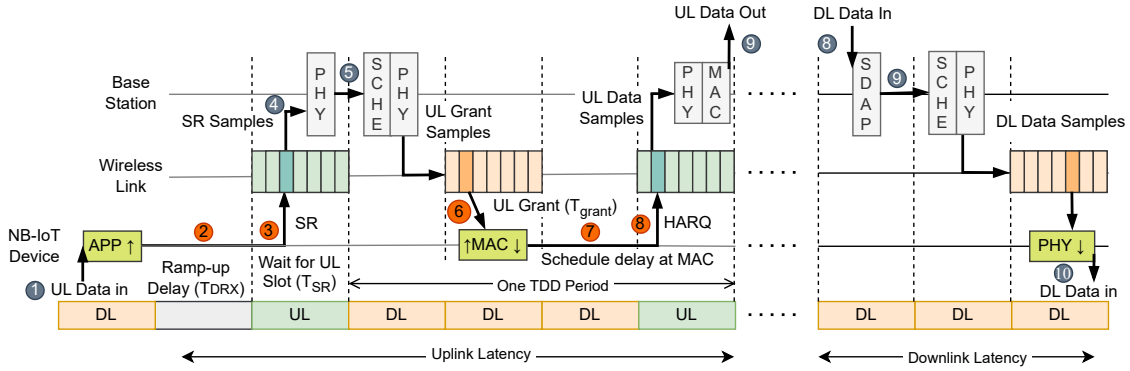



**Figure 2: NB-IoT packet transmission differs between UL and DL. In UL, the device transitions from idle to connected via the SR procedure before sending data. Both the SR and UL packets incur delays from HARQ, DL grant reception, and scheduling—roughly 2× higher than if the device were already connected. In contrast, DL packets are sent immediately after the base station issues a DL grant, avoiding idle-to-connected and MAC-layer delays on the device side.**

**Downlink Processing Delays.** In the downlink path, since the base station has a complete view of available network resources, there is no requirement to acquire a grant from the network for transmission by the base station. The base station receives DL data at Service Data Adaptation Protocol (SDAP) layer ⑧, it schedules it to the available DL resource blocks ⑨ and sends it to the user device on the next available DL slot in TDD frame ⑩. Because of high downlink demand, TDD typically allocates more slots to DL, making them more readily available than UL slots. As a result, UL transmissions often wait longer for scheduling. However, like UL, the DL path is still affected by HARQ retransmissions and DRX delays. The total DL latency would be:

$$T_{DownLink} = T_{DRX} + T_{MACSched} + T_{HARQ}$$

**Asymmetric Latency in Time Transfer Model.** Due to different processing pathways, the UL and DL latency in the 5G network is highly asymmetric. As a result, the two-way time transfer model adds a huge error in clock offset calculation. Considering $\theta$ as NTP clock offset, $t_0$, $t_1$, $t_2$, $t_3$ as device and server timestamps, and assuming that the UL and DL delays are different i.e. $\delta = T_{Prop(UL)} - T_{Prop(DL)}$, the extra error in NTP attributed to asymmetry is, $Error = \frac{T_{Prop(UL)} - T_{Prop(DL)}}{2} = \delta/2$, and therefore, $\theta = \frac{1}{2}\left[(t_1 - t_0) + (t_3 - t_2)\right] - \delta/2$. However, NTP cannot measure or correct this asymmetry, so it absorbs it as clock error, resulting in high offsets. In cellular networks, up-link latency can be up to 78% of the total Round Trip Time (RTT) of time transfer [54], accumulating time error of up to 28%. As IoT devices use inexpensive crystals (10-100 ppm), the clock drift, combined with network delay error, prevents predictable synchronization over 5G networks.

**Clock Offset and Drift under Asymmetric Delays.** A 1 ppm crystal oscillator drifts only about 1 $\mu$s per second ($\sim$ 1 ms per 1000 s), so in isolation this looks easy to correct with an occasional resync. In NB-IoT, the main problem is not the slow drift of the crystal, but that uplink/downlink (UL/DL) asymmetry makes it hard to measure and correct that drift in the first place. The estimated offset between the device clock and the reference can be written as $\theta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} + \frac{\delta_{\mathrm{UL}} - \delta_{\mathrm{DL}}}{2}$, where $\delta_{\mathrm{UL}}$ and $\delta_{\mathrm{DL}}$ are the uplink and downlink delays. In a symmetric link, $\delta_{\mathrm{UL}} \approx \delta_{\mathrm{DL}}$ and the second term is near zero. In NB-IoT, these two delays can differ by hundreds of milliseconds because of **wake-up latency**, **grant scheduling**, **retransmissions**, and **queuing**. That bias term $\frac{\delta_{\mathrm{UL}} - \delta_{\mathrm{DL}}}{2}$ is therefore orders of magnitude larger than the ppm drift (see Fig. 4). Crucially, this means that simply syncing more often does not fix the problem: the two-way exchange itself is biased by UL/DL asymmetry, so every resync can still report the wrong offset unless the asymmetry is measured or predicted. This is exactly what SynchroNB addresses.

## 4 Timing Uncertainty Analysis

The fluctuating and asymmetric nature of the cellular wireless link [66] introduces significant time synchronization errors, building upon the drift caused by the low-resolution crystals typically found in low-cost IoT hardware. Time synchronization protocols, such as NTP, are fundamentally based on the assumption of symmetric network delays for accurate time offset estimation. Our analysis shows that four cellular-specific phenomena induce high asymmetry and hence sync errors: (1) *unreliable wireless channel quality leading to retransmissions*, (2) *competition for shared wireless resources*, (3) *wake-up delays from power-saving modes*, and (4) *internal transmit queue contention in the device*. Here, we characterize these factors, identify the root cause, and highlight the challenge.

**Experimental Setup.** We used different *Hardware Platforms*[2] to conduct experiments on NB-IoT deployments with contrasting timekeeping hardware and firmware stacks. The Nordic nRF9160-DK [43] combines a 64 MHz high-accuracy crystal oscillator (HFXO) specified at 1 parts per million (ppm) with an Arm Cortex-M33 running Zephyr Real-Time Operating System (RTOS). The HFXO's tight tolerance keeps drifting below a few microseconds per second, while Zephyr's date-time API and the nRF9160's built-in LTE-M/NB-IoT modem telemetry provide accurate timestamps and network metrics. We use Zephyr's default tick-based clock as our timebase and read modem-generated GPIO timestamps directly for SNTP exchanges and latency measurements. Time is tracked using a 32-bit,

[2]The ST B-L462E-CELL1 board has an internal multispeed oscillator (MSI) factory-calibrated to only ±1% accuracy.

**Table 2: Definition of Radio-Link Metrics**

| Metric (Unit) | Definition / Formula |
|---|---|
| RSRP (dBm) | Reference Signal Received Power: average power of the reference cell signal |
| RSRQ (dB) | Reference Signal Received Quality: $(N \times RSRP)/RSSI$, N is no. of resources |
| RSSI (dBm) | Received Signal Strength Indicator: received wideband power with noise |
| SNR (dB) | Signal-to-Noise Ratio: $10\log_{10}(P_{signal}/P_{noise})$ |
| Pathloss (dB) | Path loss: attenuation between base station and device, $(P_{\text{tx,ref}} - \text{RSRP})$ |
| Tx Reps (–) | HARQ retransmissions count at the MAC layer for a given uplink PDU |

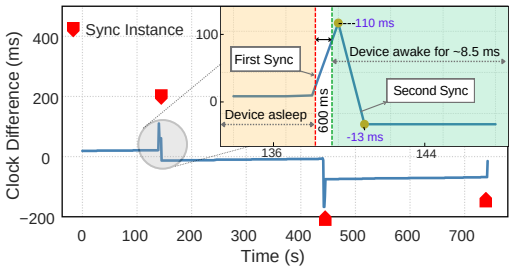



**Figure 3: During sleep, the first synchronization packet incurs a 600 ms ramp-up delay, causing a 110 ms clock offset. Upon entering connected mode, the second packet instantly corrects this offset, reducing the time difference to -13 ms.**

1 MHz hardware timer plus a software epoch offset, making this platform challenging for accurate synchronization.

*NTP and Network metrics:* We use off-the-shelf SNTP implementations compatible with the device's RTOS. For each SNTP exchange, we log the computed clock offset from the time server, the measured RTT, and all four NTP timestamp fields directly over the serial port. We also query the modem's radio link status capturing common network metrics such as RSRP, RSRQ, RSSI, SNR, and retransmission counts (Tx Reps) whose definition and details are in Table 2.

*Ground Truth:* To establish an accurate reference, we feed each device a one-pulse-per-second (1 PPS) signal to trigger a timestamp derived from the device's local time. On the Nordic, the 1 PPS pulse drives a GPIO external-interrupt (EXTI)[3]. A post-processing Python script aligns all logs to the host clock, enabling offset calculation and synchronization accuracy assessment.

*Dataset:* Across twenty-eight experiments, we collected ≈ 44190 minutes (30.6 days) worth of data, of which ≈ 40% was under good network conditions and the rest were under poor conditions.

**Sleep-State Wake-Up Latency.** NB-IoT devices employ two nested power-saving modes: eDRX and PSM to conserve battery. In eDRX, the modem periodically listens for downlink transmission in the RRC (radio resource control layer) idle state. In PSM, it enters deep sleep without periodic wake-ups. Exiting either state requires an RRC resume handshake. Our experiment (refer Figure 3) reveals a latency of ~ 600 ms (ramp-up delay) for the RRC transition to the connected state. Once connected, the device resumes its normal mode of transmission and reception.

To evaluate how PSM and eDRX affect NTP precision, we send two NTP packets while the device is in sleep mode. As shown in Figure 3, the first packet shows a 110 ms offset due to wake-up latency when the modem reconnects. The second packet, sent while the modem stays connected, reduces the offset to -13ms, correcting

[3], while on the ST B-L462E-CELL1, pps is captured via the MCU's hardware timer input-capture channel

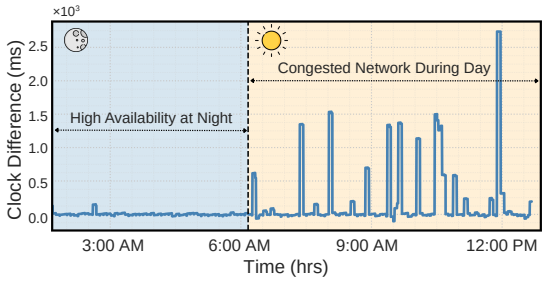



**Figure 4: High offset in daytime due to resource contention.**

the initial error. Here, 'offset' refers to the difference between the local clock and the NTP reference. We observe that the first NTP packet experiences wake-up latency during modem reactivation. However, once connected, the second packet avoids this delay, and realigns the local clock with the NTP reference.

**Challenge 1:** Power-saving wake-up delays of hundreds of milliseconds cause offset spikes during single exchanges, which NTP mistakes for clock drift, degrading accuracy after each resume.

**Resource Contention at the Base Station.** All devices within the coverage area of a single NB-IoT base station (called a "cell") share the same limited radio resources. Before an IoT node can send its NTP packet (every uplink transmission), it must issue a Scheduling Request (SR) and wait for the base station to grant uplink permission. The base station's scheduler then grants a bundle of Resource Blocks (RBs) in 1 ms Transmission Time Interval (TTI) before the device can send. When only a few devices are active (during night hours), SR→ grant (scheduling-request to grant latency) delays average around 25 ms [54], and clock offsets remain below 10 ms as shown in Figure 4. However, as more devices join in the morning (for example, students arriving on campus), the base station's scheduler is forced to arbitrate among many SRs and divide the same limited RBs among many contenders, causing grant delays to grow unpredictably into hundreds of milliseconds [34].

To quantify this effect, we place two identical NB-IoT nodes 10$m$ from the same base station and logged SR-to-grant latency and NTP clock offset every 300$s$ over a 14-hour window (00:00-14:00). As shown in Figure 4, mean offset stays around 80 $ms$ during the low-traffic period (00:00-08:00). Shortly after 09:00, when campus activity surges, the scheduler's grant delays spike, and clock differences jump up to 2.5 $s$. Because these delays occur only on the uplink, NTP attributes them entirely to clock skew, leading to large, erratic synchronization errors.

**Challenge 2:** Peak-hour resource contention inflates uplink grant delays into the 0.1-1 s range, breaking NTP's assumption of symmetric delays and causing multi-second clock errors.

**Channel-induced Retransmissions.** NB-IoT further violates asymmetry assumptions under poor signal conditions [44]. For reliability, NB-IoT uses Hybrid Automatic Repeat Request (HARQ) at the MAC layer [2]. A data packet that fails to get decoded at the receiver end is requested to be retransmitted through the HARQ process. The packet retransmission can occur in an uplink or downlink path. Nevertheless, it adds an asymmetric latency in the total RTT. Such

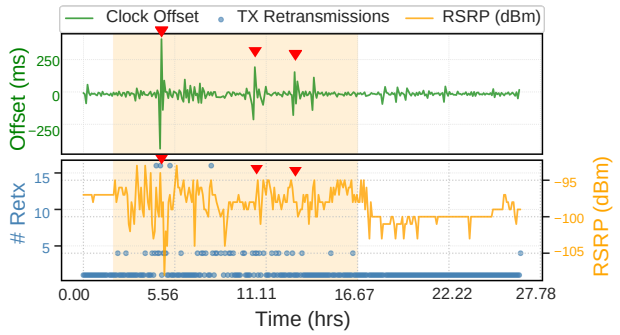



**Figure 5: Packet Retransmission due to poor signal quality.**

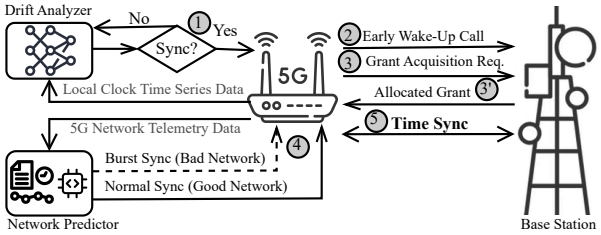



**Figure 6: SynchroNB system architecture. The clock drift analyzer triggers a time sync when needed, and the network predictor determines whether to use normal or burst sync based on network conditions.**

an asymmetry in RTT makes the NTP calculated offset inaccurate and ends up converging to the wrong time.

In Figure 5, during the relatively stable portion of the trace (0-3 h), there are no observed HARQ processes, and the NTP offset stays tightly bound around 0 ms offset. However, once the channel enters towards volatility (~ 3-16 h), the HARQ layer invokes substantially more redundancy rounds: peak retransmission counts reach 16, with a sustained mean above 4. The offset spikes to ~ 300 ms as retransmissions surges to 16. After 16 h, network conditions improve, retransmission counts subside back to 0, and the offset converges back toward its nominal envelope. The aggressive HARQ activity is a first-order driver of NTP timing error in cellular backhaul and it impacts NTP's performance quite significantly.

**Challenge 3:** HARQ undermines synchronization accuracy by introducing asymmetric retransmission delays.

**Internal FIFO Queue Contention in Modems** IoT devices typically carry both latency-sensitive traffic and bulk non-critical data streams simultaneously. Inside an NB-IoT modem, every outgoing uplink packet, whether a small NTP sync frame (48 bytes) or a larger telemetry report, is broken into RLC PDUs and placed in a single MAC-layer first-in-first-out (FIFO) buffer. Each PDU then waits its turn for the scheduler, which can only transmit one 1 ms TTI's worth of bytes (typically 32-64 B) per interval [2]. Under normal conditions, a handful of PDUs clear in a few milliseconds, and time-sync PDUs see negligible queuing delay. In a FIFO queue, if non-critical telemetry PDUs occupy the front positions, every packet behind them (including an NTP sync frame) must wait its turn, creating a *head-of-line blocking*. These internal queue delays are particularly invisible because NTP cannot distinguish device-side queueing delays from network or clock drift. NTP interprets every millisecond of delay as network latency or clock skew. Delays of tens of milliseconds at the MAC layer therefore compound any wireless asymmetries and lead to unpredictable, hard-to-correct synchronization errors.

**Challenge 4:** FIFO queue delay under mixed traffic stalls time-sync packets by 10-50 ms, injecting invisible device-internal skew that compounds network-layer errors.

**Clock Drift and Variance** Conventional low-cost IoT boards are known to ship with ±20 − 50 ppm and 32-KHz or 26-MHz crystals. For a 26 MHz crystal specified at ±40 ppm, its time offset will grow by 24 ms / 10 min.

With a 5-minute sync interval, the crystal's unchecked drift, on the order of 12 $ms$, would exceed a typical $\pm\epsilon$ $ms$ accuracy bound (for example, ±10 $ms$), thereby violating tight timing requirements and risking loss of critical functionality.

In practice, two boards with identical crystals from the same reel exhibit different drift trajectories [49]. Additionally, physical factors such as temperature, aging, and radiation level also increase the variance clock frequency, manifesting in short-term jitter and longer-term drift shifts [49]. These cannot be bounded by simple linear models due to temporal correlation. Variance creates a widening "uncertainty band" around the predicted drift line, and that band can exceed timing requirements well before the next scheduled sync. Failing to account for variance either forces frequent syncs (wasting power and bandwidth) or risks missing rapid drift shifts.

**Challenge 5:** Clock error grows both linearly (drift) and stochastically (jitter), creating an uncertainty band that can exceed synchronization accuracy requirements.

## 5 Design

In §4, we identified five independent challenges to accurate time synchronization in NB-IoT. To address them, we design SynchroNB: an on-device framework that delivers sub-10 ms synchronization within the strict energy and bandwidth constraints of NB-IoT. At its core, SynchroNB orchestrates a five-stage synchronization loop that continuously adapts to both local clock dynamics and external network variability. As shown in Figure 6, this loop is built on two complementary capabilities: first, *proactive drift analyzer* forecasts when the crystal oscillator is about to exceed its error tolerance–allowing SynchroNB to resynchronize before time sync deviates; second, *adaptive network predictor* monitors link conditions to detect when traditional single-packet synchronization may silently fail. Each sync interval begins with the MCU collecting recent clock offset estimates and radio telemetry. These inputs are fed into two compact predictors: one projecting the crystal's drift trajectory, and the other assessing wireless link stability. Their outputs jointly determine whether to initiate a standard or resilient time synchronization mechanism.

This decision process drives five tightly integrated modules. As shown in Figure 6, first the **drift analyzer** (§5.1) models the crystal drift and schedules resyncs before error bounds accumulate. When

action is needed, **adaptive wake-up scheduler** (§5.2) advances the radio boot time and injects a preemptive keep-alive ping, mitigating ramp-up latency. To ensure timely uplink access, **preemptive grant acquisition** (§5.3) issues scheduling requests just ahead of the time sync window, and the grant is allocated. When the wireless link quality degrades, the **network predictor** (§5.4) invokes burst synchronization mode. Finally, the **high-priority queue** (§5.5) elevates time-sync packets above bulk traffic to avoid contention delays while maintaining fairness.

## 5.1 Predictive Clock Drift Management

Crystal clocks produce two error sources, a deterministic offset that accumulates linearly and a stochastic jitter (§4) whose amplitude and temporal correlation differ across devices [21, 38]. A fixed resynchronization interval cannot control both effects without syncing very often, which would cost too much energy [3]. While prior time-sync protocols in GPS-enabled [40] or wired systems [17] leverage drift estimation to refine time corrections, we continuously forecast per-device drift on constrained hardware. In doing so, we implement a two-phase on-device module: first, we select and preprocess features directly tied to oscillator behavior; then, we train and deploy a compact model to forecast imminent drift and *decide when to sync*.

*5.1.1 Feature Selection.* To reliably forecast crystal drift and avoid needless synchronizations, we distilled the inputs to a few signals that meet our design principles: First, **direct physical relevance** requires us to utilize the local clock data i.e. the absolute difference between locally elapsed time and true time to capture drift behavior. The offset $x_t$ is the absolute difference between the locally elapsed time and the corresponding true reference time at measurement instance, given by: $x_t = |t_{\text{local},t} - t_{\text{true},t}|$. This offset directly quantifies the instantaneous deviation of the local clock from the reference time. **Low-overhead observability** requires keeping cost and power draw minimal. $x_t$ is obtained from existing firmware telemetry. $t_{\text{true},t}$ is recorded by the reference device connected to the MCU. $t_{\text{local},t}$ is logged by the MCU's built-in clock. These measurements do not require additional sensors, extra telemetry, and complex cross-layer counters, utilizing a few microseconds of MCU time per sample and consuming only a few dozen bytes of RAM.

*Drift Factors.* Crystal stability can also be influenced by external and intrinsic factors such as temperature, supply voltage, phase noise, and aging. While these parameters contribute to long-term drift, most commercial NB-IoT modules do not expose such telemetry through standard interfaces, and adding dedicated sensors would increase cost and energy overhead. To preserve portability across hardware, SynchroNB focuses on a *minimal, device-only feature set* derived from offset history. This design choice keeps the model deployable on low-cost NB-IoT devices while remaining compatible with optional external inputs, which enhance prediction accuracy without modifying the control loop.

*5.1.2 Model Selection.* The local clock error has two parts: a linear drift and a random jitter component. In practice, the short-term changes in this error are very noisy and not purely random. This behavior is shaped by network and device conditions and persists over multiple sync cycles. On the network side, factors such as paging delay, uplink grant scheduling, retransmissions, and queuing can introduce sudden one-way timing jumps; when the cell is congested or signal quality is poor, these errors can repeat for several successive syncs. On the device side, temperature and supply variation can subtly change oscillator stability over time. Traditional drift estimators, such as regression and Kalman filtering, assume stable jitter and require frequent, fixed-interval timestamp updates. They model smooth drift well but do not capture noisy jitter, and they perform poorly when samples are sparse. Because NB-IoT exhibits both long-term drift and short-lived jitter patterns that depend on recent history, we treat drift prediction as a sequence forecasting problem. Given a short history of recent offset samples, we predict the future error and trigger synchronization only if the predicted error is about to exceed the required accuracy.

Simple linear models and ARIMA [9, 42] can capture the long-term drift trend, but they assume that the noise is stable over time. As a result, they miss device-specific timing patterns and react slowly when the noise changes. Kalman-based estimators and their variants [26, 28] work well when the process and measurement noise are well modeled and when updates arrive often, but they require careful tuning that does not transfer across hardware or network conditions. They also lose accuracy when the timing samples are sparse or irregular. Statistical machine learning approaches treat clock drift and jitter as random variables and update their estimates using likelihood models learned from past observations [62]. These methods can improve robustness but they still assume frequent timestamp exchanges and well-characterized link delays. These constraints point to a recurrent model that can learn how drift evolves over time, suppress noise bursts, and update its internal state even when samples arrive at irregular intervals.

We choose a single-layer Long Short-Term Memory (LSTM) [23] because it can use recent history to recognize both slow drift and short-lived jitter bursts, does not require regular timestamps, and remains small enough to run on the device. An LSTM maintains an internal state (the memory cell) that lets it track slow drift over time while also modeling short-lived bursts of jitter, rather than smoothing them away. Its gating mechanism also lets it selectively "forget" outdated behavior when conditions change (for example, due to temperature or supply variation) without us having to manually retune process noise. Unlike linear or Kalman-based estimators, the LSTM does not assume fixed noise statistics or require regular sampling; it learns the device's own offset dynamics directly from recent history and produces a prediction of future error.

At time $t$, the model consumes a short window of normalized offset samples $\mathbf{x}_t = [x_{t-N+1}^{\text{norm}}, \ldots, x_t^{\text{norm}}]$ and outputs a scalar forecast $\Delta_{\text{pred}}$. A synchronization is triggered *only* when $\Delta_{\text{pred}} > \epsilon$.

*Online Adaptation.* To accommodate gradual aging or slow environmental shifts, SynchroNB supports lightweight online adaptation. To ensure responsiveness to long-term drift trends, the model parameters are periodically updated using an exponential moving average over recent offsets. This process runs infrequently, only once a day, and adds negligible overhead on the device.

Our drift analyzer combines streamlined feature selection with a compact LSTM model, enabling NB-IoT devices to predict clock drift accurately. By triggering synchronization only when needed,

**Algorithm 1** Adaptive Wake-Up Scheduling

**Require:** Histogram $H$, energy cap $E_{\max}$, guardrail $\alpha$, next sync time $t_{\text{sync}}$
$\Delta \leftarrow$ Percentile($H$, 95%)
SetTimer($t_{\text{sync}} - \Delta$, WakeAndSync())
**function** WakeAndSync
  $d \leftarrow$ ModemWakeLatency() ▷ RRC resume delay
  Insert($H$, $d$)
  **if** EnergyUsed $> \alpha E_{\max}$ **then**
    PerformSync
  **else**
    SetTimer($t_{\text{sync}}$, PerformSync())
  **end if**
**end function**

our approach reduces energy and bandwidth usage without compromising time accuracy.

## 5.2 Adaptive Wake-Up Scheduling

After forecasting when the local clock will exceed its drift budget, in this section, we bound the RTT variance caused by large modem wake-up delays (as discussed in §4). NB-IoT devices conserve power by entering eDRX or PSM sleep modes, but waking the modem adds on the order of 100 $ms$ of one-way RTT variance (§4). A naive fix is to hard-code a long lead time ($\Delta$ seconds) before each sync. In theory, a fixed lead time, for instance, $\Delta = 3s$ would cover every modem wake-up and eliminate the cold-start penalty. In practice, however, any single static margin is either wastefully long or woefully insufficient. Due to **variable wake-up latency**, the modem attach delays fluctuate with cell-edge reselection, tracking-area updates, and neighbor scans [54]. Any $\Delta$ large enough to cover the worst-case transition will be wastefully long most of the time.

To balance these, in SynchroNB we employ a feedback-driven adaptive wake-up scheduler (captured in Algorithm 1). The scheduler maintains a latency histogram $H$ to track recent wake-up delays. For each synchronization event, the device sets a coarse timer ($\Delta_{(coarse)}$) based on the 95 percentile latency derived from $H$, waking only as early as experience demands. This percentile-based margin automatically adapts to prevailing conditions: if wake-up delays grow, $\Delta_{\text{coarse}}$ increases; if they shrink, they decrease. Upon waking (`WakeAndSync()`), the modem's actual wake-up latency (RRC resume delay) is measured and immediately used to update the histogram, ensuring the percentile estimate remains up to date.

To prevent excessive energy consumption, the scheduler checks whether the radio-idle energy consumed has exceeded a predefined fraction $\alpha$ of the synchronization energy budget ($E_{\max}$). If this threshold is breached, an immediate synchronization is performed, effectively limiting energy usage. By continuously updating $H$, re-computing $\Delta_{\text{coarse}}$, and enforcing an energy guardrail, this self-tuning loop provides short wake-up time bounds while consuming only the energy necessary under prevailing network conditions.

## 5.3 Preemptive Resource Allocation

SynchroNB next tackles the remaining source of asymmetric delays i.e. the uplink scheduling delays under network contention. In §4, we showed that in NB-IoT, each synchronization packet must first issue an SR and await a grant before it can transmit. Under light load, this handshake completes in a few milliseconds [54], but as cell load grows, $SR \rightarrow grant$ delays grow causing NTP to misattribute those delays to clock skew. A simple *always ask early* policy (sending every SR one or two SR-periods ahead) either wastes scarce RBs when the cell is idle or, if every device does it, amplifies contention and collapses fairness. Instead, in SynchroNB we introduce **an adaptive, two-phase closed-loop reservation scheme** that pulls airtime only when contention is predicted:

**(1) Contention estimation.** At each sync interval, the network predictor (§5.4) computes a short-horizon *grant-delay risk* from recent radio metrics (RSRP, RSRQ, SNR, Tx Reps, time of day). We discuss this more in §5.4. If this risk exceeds a tunable threshold $\phi$, which we tune empirically, the upcoming sync slot is flagged as *contention-likely*.

**(2) Just-in-time SR advance.** Only when a slot is marked *contention-likely* does the modem issue its SR exactly one SR period (5-10 ms [54]) before the scheduled sync. This ensures that the uplink grant arrives immediately before the sync packet, avoiding both idle RBs and SR blocking.

After each synchronization, SynchroNB observes the actual $SR \rightarrow grant$ delay; if it still exceeds one SR period, we decrease $\phi$ (making future early-SR decisions more frequent); if it consistently falls within a single period, we increase $\phi$ (deferring early SRs). This feedback loop automatically tracks both daily load patterns (predictable) and transient hotspots without manual tuning. It preserves fairness because slots not marked as high contention follow the standard SR timing, so all devices continue to share the scheduler equitably. It reduces worst-case $SR \rightarrow grant$ to a single SR cycle, reducing synchronization errors by orders of magnitude [54].

## 5.4 Volatility-Driven Burst Synchronization

SynchroNB now targets the asymmetric delays introduced by HARQ retransmissions as explained in §4. Standard SNTP exchanges treat any extra uplink delay as clock skew. To counteract this, we recast sync timing as a classification problem, predicting when link conditions will violate our symmetric-delay assumption (we call it "Network Predictor"). When such volatility is detected, SynchroNB switches to resilient-sync mode, which we later explain.

*5.4.1 Feature Selection:* To reliably flag when asymmetric link delays will break single-packet synchronization, we choose features that satisfy the same design principles used in the drift analyzer. To provide **direct physical relevance**, we monitor six radio-layer metrics whose behaviors directly drive retransmissions and scheduler delays. RSRP and RSRQ quantify signal strength and quality, SNR measures noise margin, path loss reflects propagation conditions, uplink repetition count indicates the configured HARQ redundancy (and thus expected retransmissions), and time of day serves as a coarse proxy for diurnal cell load variations. Together, these features capture the principal factors driving retransmissions and scheduler delays.

For **low-overhead observability**, all metrics are natively exposed by the modem's standard attention telemetry interface. Sampling them every sync interval requires no extra sensors, no firmware modifications, and only $\sim 200\mu s$ of MCU time plus a few dozen bytes of RAM for rolling buffers [14]. Essentially, **stream orthogonality** should prevent both predictors' cross-contamination. To do so, we exclude any clock error or latency values from this feature set. By keeping the network predictor blind to clock behavior, we prevent

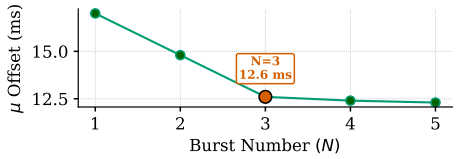



**Figure 7: Mean clock offset (ms) as a function of burst number $N$. Most of the reduction, down to 12.6 $ms$, occurs by $N = 3$ (highlighted), with further packets yielding marginal gains.**

information leakage, while focusing solely on link dynamics, and not conflate drift events with network anomalies.

*5.4.2 Model Selection.* We need a classifier that recognizes temporal micro-bursts in the six-dimensional feature stream and executes in under a few milliseconds on our MCU, to predict network volatility. After evaluating several candidates, we analyzed gradient-boosted trees [10], static convolution neural network (CNN) [32], recurrent neural network (RNN) [23], and clustering methods like K-means [36] or Gaussian Mixture Models [13], but each fell short, trees and clustering ignore temporal order [8], static CNNs require extensive feature engineering [32], and RNNs incur unacceptable latency and memory overhead [24]. These observations led us to adopt a causal **Temporal Convolutional Network (TCN)**.

The TCN uses causal, dilated 1-D convolutions to learn both short- and mid-range patterns that come before asymmetric delays such as retransmissions or scheduler stalls. This lets it model transient link behavior while ensuring that each prediction depends only on past observations. Our implementation has three layers with kernel size 3 and dilations of 1, 2, and 4, and it fits in under 8 KiB of flash and less than 1 KiB of RAM. The model is trained offline on labeled synchronization outcomes from different link conditions and outputs a probability $p$ that the link is "volatile." When $p$ exceeds a threshold $\phi$, the device switches to burst synchronization. This design keeps inference causal, lightweight, and easy to interpret, so it can run on NB-IoT hardware.

*5.4.3 Resilient Synchronization Strategy.* We implement a resilient-sync strategy to mitigate the unpredictable, uplink-only delay spikes caused by HARQ retransmissions. Transmitting a burst as $N$ consecutive SNTP requests instead of a single packet, SynchroNB increases the success rate. The resultant offset with the smallest absolute value is chosen, on the premise that it encountered the least network delay.

At each sync interval, we first compute the link-volatility score $p$; if $p > \phi$ (empirically set to 0.3 to balance false positives and missed detections), we launch a burst of $N$ requests, spacing them by one 1 ms TTI plus a 2 ms processing gap to avoid internal modem state collisions. We then gather the $n$ offsets $o_1, o_2, \ldots, o_n$ and apply a minimum absolute effectively filtering out outliers caused by HARQ loops or grant tail delays. Through our experiments shown in Figure 7, we set $N = 3$ as a reasonable compromise between sync accuracy and higher power and bandwidth consumption.

## 5.5 Priority-Based MAC Queuing

After fixing all the other sources of asymmetrical delay, SynchroNB fixes MAC queuing delay caused by intra-device contention discussed in §4. Traditional network-side remedies, like dedicating a 5G slice or a high-QoS bearer [20], impose heavy RRC signaling and can still be pre-empted by other high-priority flows. Rather than creating new radio bearers or slices, which incur extra RRC signaling and still risk preemption [20], we implement fine-grained prioritization entirely inside the existing Data Traffic Channel (DTCH). We split the DTCH into configurable number of "n" logical queues inside the MAC layers; a *High-Priority Queue (HP-Q)* for time-sync frames and a *Best-Effort Queues (BE-Q)* for all other traffic. When the application constructs an SNTP request, it tags the packet for HP-Q insertion. During each *SR*, the modem reports HP-Q occupancy before BE-Q in its buffer-status reports (BSRs), the very next uplink grant clears one HP-Q PDU, before any BE-Q traffic is scheduled. This prioritization reduces internal queuing delays and ensures deterministic priority for time-sync packets. If the network is stable, a lone time-sync PDU enters HP-Q, bypasses any backlog in BE-Q, and is scheduled in the very next TTI, bypassing all other application data with higher scheduling priority.

**Performance under degraded conditions.** Under volatile network, the network predictor suggests a burst sync, i.e., $N$ consecutive SNTP frames. Flooding HP-Q with all $N$ PDUs would starve other applications of resources. To ensure fairness, we admit only the first PDU of each burst into HP-Q; the remaining PDUs enter BE-Q. Each PDU carries two timers in MAC metadata: an *aging window ($T_{hp}$)*, that tracks how long the PDU waits at the head of HP-Q (one SR period, ~ 5-10 ms [54]). A *slack window* ($T_S$), that measures time in BE-Q (two grant intervals, ~15 ms [54]).

• **HP-Q.** While the $T_{\text{hp}}$ has not expired, the packet stays in HP-Q and is immediately scheduled upon grant availability. If $T_{\text{hp}}$ elapses, because another PDU pre-empted the grant, the packet climbs down into BE-Q, resetting its timers.

• **BE-Q.** When a BE-Q PDU's slack timer $T_S$, expires, it re-promotes into HP-Q. This alternating promotion/demotion guarantees that exactly one sync PDU occupies the high-priority path at a time while naturally interleaving burst traffic with normal payloads.

## 5.6 SynchroNB: Putting it all together

At each interval, the control loop of SynchroNB first examines recent clock offsets and uptime in **drift analyzer** module and predicts whether the clock's accumulated error will breach our set threshold, if not, the loop can defer syncing; otherwise, it triggers a sync. Once a sync is required, our **adaptive wakeup scheduler** uses its learned histogram to rouse the modem just in time and injects a short keep-alive ping to eliminate the unpredictable RRC handshake tail. Next, **preemptive SR** module requests precisely one SR period before the SNTP packet, that the uplink grant lands immediately when the time message goes out. As the SNTP exchange proceeds, our **network predictor** simultaneously evaluates link volatility; if it flags abnormal HARQ or scheduler delays, the loop switches into resilient burst sync mode. Finally, at every transmission, the **high-priority queue** guarantees that each sync frame, whether single or burst, clears the MAC's FIFO ahead of bulk telemetry, reducing the MAC queuing delays.

# 6 Implementation

We implement SynchroNB on the `nRF9160` IoT board running the Zephyr RTOS. All the traces, GPIO event logs, clock, and network metrics are captured using the same experimental setup listed in

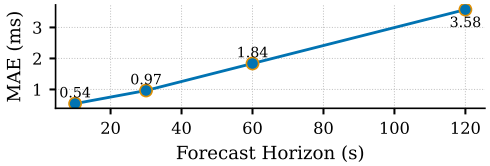



**Figure 8: Mean Absolute Error (MAE) of the LSTM drift predictor as a function of the forecast horizon. The model achieves sub-millisecond accuracy for short-term (10s, 30s) forecasts and degrades to an error of under 4 ms at a 120s horizon.**

§4. The SynchroNB software stack comprises of two primary modules. (1) A synchronization manager, running as a Zephyr thread, captures the wake-up timers, drift checks, scheduling-request injections, and SNTP exchanges. (2) Two on-device machine-learning predictors on TensorFlow Lite Micro [57] for the efficient and lightweight execution of LSTM for drift prediction and a TCN for network-volatility classification. The predictive models are fully quantized to 8-bit integers and execute without floating-point hardware or dynamic memory allocation. Each inference requires only a few hundred multiply-accumulate operations, the LSTM occupies $\sim 6.8kB$ of flash memory, and the TCN $\sim 9kB$, well within the limits of typical NB-IoT MCUs.

We trained our models on the dataset described in §4. This dataset is large and diverse, enabling rapid convergence and yielding very high test accuracies. Inference runs entirely on the board's Cortex-M3, with no dynamic memory allocation or floating-point support required. LSTM and TCN run on parallel threads, invoked by the synchronization manager at each scheduled interval, and neither model stalls the radio stack or jeopardizes real-time deadlines.

To keep both models up to date without overburdening the device, we adopt a hybrid update strategy. After each sync, a lightweight bias correction adjusts the LSTM's output based on the latest prediction error, and the TCN's volatility threshold is shifted by a small factor towards the observed grant delays. Periodically, once per day or when the device is charging, we download a quantized weight patch for the TCN over the cellular link, applying it in place to refresh its parameters. Meanwhile, the LSTM receives an optional one-step gradient update using the most recent offset history for fast, in-system backpropagation. This continual, on-device adaptation keeps both predictors tightly aligned with evolving oscillator behavior and network dynamics, all while consuming only a few hundred microjoules per patch.

The **drift predictor** is a lightweight two-layer recurrent network designed to run on constrained IoT hardware. It consumes a $10 \times 1$ input vector of the most recent clock-offset measurements and outputs a one-step ahead estimate of drift. A single LSTM layer with 32 hidden units first processes the time series, compressing both the slow, linear drift and faster, stochastic jitter into a 32-dimensional hidden state. A final fully connected output layer then projects this embedding to a single scalar ($\Delta_{pred}$), representing the predicted drift over the next interval. We evaluate the drift predictor's short-term accuracy in Figure 8, when forecasting 10-30 s ahead, this model routinely achieves sub-millisecond error. By comparing $\Delta_{pred}$ against our threshold $\epsilon$, the synchronization manager can confidently schedule resynchronization only when the clock is forecast to exceed its allowable error.

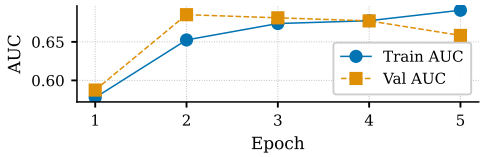



**Figure 9: Training and validation AUC of the temporal convolutional network over five epochs.**

We implement our **network predictor** as a TCN that classifies each upcoming sync slot as stable or volatile, driving our choice of single-packet versus resilient burst synchronization strategy (see § 5.4). TCN ingests a small window of recent network measurements arranged as a $6 \times 4$ tensor (six metrics over four-time steps). Its three TCN blocks each apply a causal 1D convolution (16 filters, kernel size 3) with dilations of 1, 2, and 4 respectively, followed by batch normalization and a residual shortcut to preserve fine-grained temporal context. A global average-pooling layer reduces the resulting $6 \times 16$ feature map to a 16-dimensional vector, which a final fully-connected layer projects to a single probability $p$ of volatile. In Figure 9 we show, during offline training, our TCN quickly converges, hitting a validation AUC of 0.68 by epoch 2, further epochs yield only marginal gains and begin to overfit.

Overall, our SynchroNB implementation emphasizes minimal resource utilization, tight integration with existing IoT platform features, and robust, real-time responsiveness to maintain accurate synchronization in practical cellular IoT deployments.

## 7 Evaluation

We evaluate the effectiveness of SynchroNB across several dimensions critical to real-world cellular IoT deployment. Our evaluation study answers the following key questions:
(Q1) How does SynchroNB perform compared to SNTP?
(Q2) How does SynchroNB maintain robust performance across varying cellular network conditions?
(Q3) How do the individual components of SynchroNB contribute to improved time synchronization?
(Q4) What is the overhead of SynchroNB on resource-constrained cellular-IoT devices?

We find that SynchroNB cuts worst-case synchronization error from 257 *ms* to 5 *ms*, and maintains robust performance under degraded network. Importantly, with minimal energy overhead and bandwidth utilization, SynchroNB achieves reliable and consistent sub-10ms clock synchronization over the cellular network.

**Baseline.** We use SNTP as the primary quantitative baseline because it represents the *vendor-supported, deployable* synchronization option available on NB-IoT modules today. SNTP operates entirely at the device side and requires no base-station or core-network modifications, which aligns with the scope of SynchroNB's device-only design. For fairness, all experiment conditions were evaluated under identical hardware platforms, polling frequency, time stamping paths, and modem power states.

**Clock Accuracy.** Accurate synchronization without exchanging frequent two-way sync packets is crucial for low-power IoT devices. To evaluate how effectively SynchroNB meets this challenge, we benchmark its accuracy against the conventional SNTP protocol in a controlled 6-hour experiment, comparing the frequency of sync events and the resulting offsets.

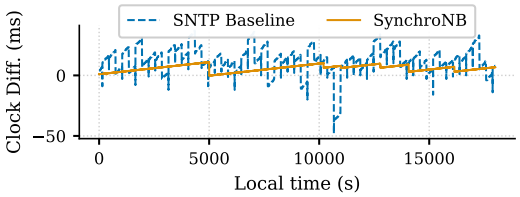



**Figure 10: Comparison of synchronization accuracy between SynchroNB and the standard SNTP baseline.**

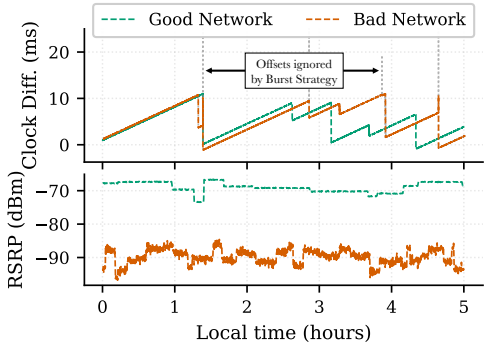



**Figure 11: Synchronization accuracy of SynchroNB under good versus poor network conditions represented as RSRP.**

As shown in Figure 10, the first sync under our system occurs around 1.38 hours into the experiment. Notably, the second, third, and fourth syncs occur in relatively quick succession. The cluster of syncs shortly after the initial synchronization indicates an adaptive correction process, triggered by slight residual offset that was quickly countered. This behavior underscores two strengths of our system: (1) the LSTM-based drift analyzer effectively suppresses unnecessary syncs under low drift conditions, and (2) when an offset is not fully corrected in a single sync, the drift analyzer quickly detects the residual deviation and triggers follow-up corrections before the offset spikes.

Despite significantly fewer syncs, our method maintains tight synchronization with a mean clock difference of $\approx 4ms$ between the two boards over the full 6-hour run, compared to SNTP's 14.78 $ms$. Note that this is a result during good network conditions. This shows that accurate timekeeping can be maintained without frequent network communication, highlighting the synergy between our drift-aware scheduling and adaptive sync strategy.

**Network Variations.** During good and poor cellular network quality, represented through different RSRP values, Figure 11 illustrates that our solution consistently maintains stable synchronization across both network states. Over a continuous 12-hour experiment, 6 hours under good network conditions and 6 hours under degraded network, we observed mean clock differences of 4.6 $ms$ and 5.3 $ms$ respectively. Notably, the adaptive network prediction mechanism with the burst synchronization strategy significantly enhanced synchronization stability during periods of poor network quality. The network predictor effectively identifies episodes of network volatility, prompting the devices to initiate burst synchronization. Consequently, erroneous high-offset measurements from initial packets were discarded, ensuring tighter overall synchronization.

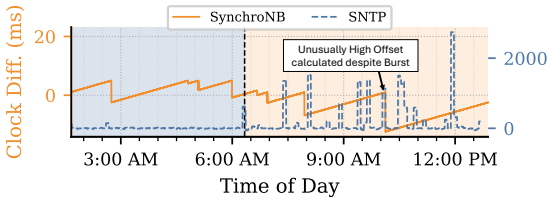



**Figure 12: Clock-difference traces over the course of a night-to-day experiment (note the different y-axis), comparing SynchroNB (solid orange, left-axis) with the SNTP baseline (dashed blue, right-axis). A vertical dashed line at 6:20 AM separates two phases: daytime (orange shading) and nighttime (blue shading).**

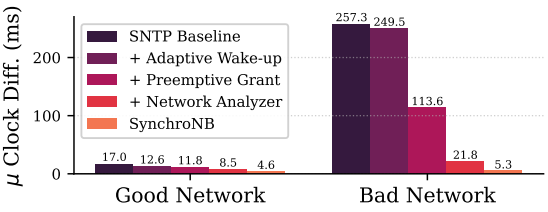



**Figure 13: Average clock-difference (ms) for five synchronization strategies: SNTP Baseline, Adaptive Wake-up, Preemptive Grant, Network Predictor-based Burst, and SynchroNB under both Good and Bad network conditions.**

We further assess synchronization performance under varying network loads by comparing day and night conditions. Figure 12 presents results from a long-duration experiment capturing the performance of SynchroNB compared to standard SNTP. During, nighttime, both approaches exhibit long-term stability, although SynchroNB maintains significantly tighter synchronization, achieving an average offset of 4.49 $ms$ compared to SNTP's 83.8 $ms$. In contrast, during daytime peak usage hours, SNTP experiences substantial performance degradation due to increased network contention. While SynchroNB also sees a slight reduction in synchronization accuracy, the utilization of preemptive grant requests and packet prioritization mechanisms effectively mitigates the negative impacts of high contention. Consequently, SynchroNB maintains a mean offset of only 6.22 $ms$, whereas SNTP's offset dramatically rises to 154.3 $ms$ with a $95^{th}$ percentile exceeding 1.3 seconds. A key insight is that the drift analyzer within SynchroNB triggers synchronization events strictly when necessary, significantly reducing the frequency of synchronization compared to standard interval-based protocols. This intentional design minimizes the risk of significant clock drift accumulation.

**Ablation Study.** To understand the effect of individual contributions and effectiveness of each component of SynchroNB, we perform a series of ablation studies. These experiments systematically remove or isolate specific mechanisms within our synchronization solution to quantify their individual impact on accuracy under varying network conditions. Specifically, we evaluate four key strategies: (1) Adaptive Wake-up, (2) Preemptive Grant, (3) Network Predictor-based burst, and the complete SynchroNB approach, against the baseline SNTP protocol.

Figure 13 compares the average clock difference from long-running experiments under both good and bad network conditions.

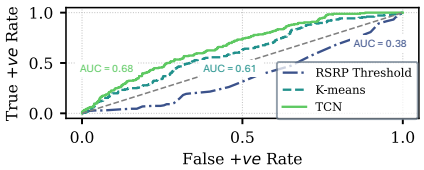
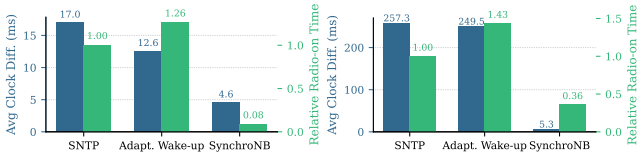



**Figure 14: Receiver operating characteristic (ROC) curves comparing three network-condition prediction methods: RSRP thresholding, unsupervised K-means clustering, and a temporal convolutional network. The diagonal dashed line indicates random guessing.**




**(a) Good network** **(b) Bad network**

**Figure 15: Energy-accuracy trade-off for three synchronization strategies (SNTP baseline, adaptive wake-up, and SynchroNB) under (a) good and (b) bad cellular network conditions. SynchroNB achieves the lowest average error while reducing radio-on time in both scenarios.**

The SNTP baseline exhibits significant errors, averaging a clock difference of 17.0 *ms* under good conditions and dramatically worsening to 257.3 *ms* under adverse conditions. Incorporating Adaptive Wake-up alone slightly improves accuracy (12.6 *ms* in good conditions, 249.5 *ms* in bad conditions). Adding the Preemptive Grant further reduces offsets to 11.8 *ms* (good) and 113.6 *ms* (bad). The introduction of the Network Analyzer significantly enhances performance, achieving mean offsets of 8.5 *ms* (good) and 21.8 *ms* (bad). The full SynchroNB approach, which combines all components, yields the lowest average offset of 4.6 *ms* under good network conditions and maintains performance at 5.3 *ms* under bad conditions. These results highlight the essential roles of adaptive wake-up, proactive grant requests, and network prediction in maintaining robust synchronization performance across varying network states.

We further evaluate the predictive accuracy of our network predictor by comparing its performance with baseline methods. Figure 14 presents the Receiver operating characteristic (ROC) curves for three distinct prediction approaches: (1) a naive threshold-based using RSRP (we use RSRP $\leq -90$ as blanket threshold), (2) an unsupervised K-means clustering baseline, and our TCN-based predictor. The RSRP threshold-based method achieves an AUC of 0.38, indicating poor discriminative ability. K-means clustering provides a substantial improvement, achieving an AUC of 0.61, demonstrating that even basic unsupervised learning can offer meaningful predictions. However, our TCN model outperforms both baselines, reaching an AUC of 0.68, highlighting its capability to accurately classify network conditions. These results underscore the effectiveness of employing a specialized temporal model like TCN for robust network condition predictions in dynamic environments.

**Overhead Analysis.** We evaluate the overhead introduced by SynchroNB in terms of *energy consumption* and *bandwidth utilization*.

Figure 15 shows the energy-accuracy trade-off under both good and bad network conditions. Under favorable network conditions

**Table 3: Bandwidth overhead (sync message count) and mean offset under Good/Bad network conditions.**

| Strategy | # Syncs (Good/Bad) | Mean Offset (Good/Bad) [ms] |
|---|---|---|
| SNTP Baseline | 120 / 120 | 16.98 / 257.26 |
| Adaptive Wake-up | 120 / 120 | 12.55 / 249.47 |
| Pre-emptive Grant | 120 / 120 | 11.85 / 113.63 |
| NP-based Burst | 135 / 165 | 8.50 / 21.77 |
| SynchroNB | 7 / 30 | 4.55 / 5.27 |

(Figure 15a), SynchroNB achieves an average clock difference of 4.6 *ms*, performing better than the SNTP baselines (17.0 *ms*) and the adaptive wake-up strategy (12.6 *ms*). Based on the power profile reported in the nRF9160 datasheet[4], we estimate that each synchronization exchange consumes approximately 150 mJ, which includes radio transmission, reception, and modem wake-up energy. Over the six-hour experiment, this corresponds to a total energy of 1.05 J under good conditions (7 exchanges) and 4.5 J under poor conditions (30 exchanges). Crucially, our approach achieves this accuracy with substantially lower energy overhead, reducing the total synchronization energy to roughly 8% of the SNTP baseline. Even under degraded conditions (Figure 15b), the adaptive nature of SynchroNB mitigates sync errors (mean offset of 5.3 *ms* compared to 257.3 *ms* for SNTP), while maintaining 36% of the baseline energy despite using burst synchronization to handle network volatility.

Table 3 shows bandwidth overhead through message count comparisons. Over a 6-hour experiment, the standard SNTP and adaptive wake-up approaches transmit 120 synchronization packets each, irrespective of network conditions. In contrast, SynchroNB's intelligent synchronization strategy reduces bandwidth consumption, requiring only 7 sync packets in good network conditions and 30 in bad conditions. This adaptive reduction directly minimizes network resource usage, highlighting SynchroNB's practical advantages for large-scale NB-IoT deployments.

*Scope and Extensions.* Our predictor handles short-term, time-varying offset behavior caused by NB-IoT communication. In deployments with large environmental changes or very long lifetimes, we can also include external signals such as temperature or supply voltage to improve accuracy. These additions do not require any changes to the control loop and fit within the current design.

Our evaluation uses a controlled testbed to isolate NB-IoT timing behavior under repeatable conditions. We acknowledge that we do not evaluate urban or rural field trials, high-mobility handovers, persistent connectivity loss, or multi-cell or multi-hop synchronization. Still, the timing behavior and uplink/downlink asymmetry we observe form the basis for extending SynchroNB to those settings.

## 8 Conclusion

This paper addresses the critical challenge of achieving reliable, sub 10-ms time synchronization on NB-IoT devices under unreliable network conditions. Our preliminary analysis exposes the fundamental causes of time errors resulting from highly asymmetric cellular network delays that render conventional protocols like NTP ineffective. We propose SynchroNB, an on-device, ML-powered framework that anticipates and compensates for both network volatility and clock drift. By dynamically adjusting modem wake-ups, and uplink

[4]Energy estimates are derived from the nRF9160 datasheet's measured transmission and reception current profiles, assuming a nominal 3.7 V supply.

resource requests, SynchroNB transforms time sync from a passive polling task into an intelligent, adaptive control loop.

## Acknowledgments

We thank our shepherd and the anonymous reviewers of ACM SenSys 2026 for their insightful feedback and constructive suggestions, which greatly improved the clarity and quality of this paper. This work was supported by the National Science Foundation under Grant numbers 2237485 and 2230143.

## References

[1] 3GPP TSG SA WG1-Services. [n. d.]. *3GPP TS 22.042: Network Identity and Time-Zone (NITZ); Service description; Stage 1.* Technical Specification TS 22.042. 3rd Generation Partnership Project. Stage 1 service description for NITZ.

[2] 3rd Generation Partnership Project (3GPP). 2019. *TS 36.321: Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification.* Technical Specification TS 36.321. 3GPP. https://www.3gpp.org/DynaReport/36321.htm ETSI TS 136 321 V15.7.0 (Release 15).

[3] Anton Ageev. 2010. *Time synchronization and energy efficiency in wireless sensor networks.* Ph. D. Dissertation. University of Trento.

[4] Shilpy Agrawal, Khvati Chopra, et al. 2022. Analysis of energy efficient narrowband internet of things (NB-IoT): LPWAN comparison, challenges, and opportunities. *Wireless Communication with Artificial Intelligence* (2022), 197–217.

[5] Muhammad Akhlaq and Tarek R Sheltami. 2013. RTSP: An accurate and energy-efficient protocol for clock synchronization in WSNs. *IEEE Transactions on Instrumentation and Measurement* 62, 3 (2013), 578–589.

[6] Farzad Asgarian and Khalil Najafi. 2021. BlueSync: Time synchronization in Bluetooth low energy with energy-efficient calculations. *IEEE Internet of Things Journal* 9, 11 (2021), 8633–8645.

[7] Mir Ali Rezazadeh Baee, Leonie Simpson, Ernest Foo, and Josef Pieprzyk. 2019. Broadcast authentication in latency-critical applications: On the efficiency of IEEE 1609.2. *IEEE Transactions on Vehicular Technology* 68, 12 (2019), 11577–11587.

[8] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery* 31 (2017), 606–660.

[9] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control.* John Wiley & Sons.

[10] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[11] Nikumani Choudhury, Rakesh Matam, Mithun Mukherjee, and Lei Shu. 2018. Beacon synchronization and duty-cycling in IEEE 802.15. 4 cluster-tree networks: A review. *IEEE internet of things journal* 5, 3 (2018), 1765–1788.

[12] Dennis Cox, Emil Jovanov, and Aleksandar Milenkovic. 2005. Time synchronization for ZigBee networks. In *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory, 2005. SSST'05.* IEEE, 135–138.

[13] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society: series B (methodological)* 39, 1 (1977), 1–22.

[14] Almudena Díaz-Zayas, Cesar A García-Pérez, Alvaro M Recio-Pérez, and Pedro Merino. 2016. 3GPP standards to deliver LTE connectivity for IoT. In *2016 IEEE first international conference on internet-of-things design and implementation (IoTDI).* IEEE, 283–288.

[15] Benjamin Dowling, Douglas Stebila, and Greg Zaverucha. 2016. Authenticated network time synchronization. In *25th USENIX security symposium (USENIX security 16).* 823–840.

[16] Marco F Duarte, Michael B Wakin, Dror Baron, and Richard G Baraniuk. 2006. Universal distributed sensing via random projections. In *Proceedings of the 5th international conference on Information processing in sensor networks.* 177–185.

[17] John C Eidson, Mike Fischer, and Joe White. 2002. IEEE-1588™ Standard for a precision clock synchronization protocol for networked measurement and control systems. In *Proceedings of the 34th annual precise time and time interval systems and applications meeting.* 243–254.

[18] Mahmoud Elsharief, Ahmed Emran, Hossam Hassan, Saifur Rahman Sabuj, and Han-Shin Jo. 2024. Energy-Efficient Synchronization in Industrial Internet of Things: An Intelligent Neighbor-Knowledge Approach. *IEEE Transactions on Industrial Informatics* (2024).

[19] Jeremy Elson, Lewis Girod, and Deborah Estrin. 2002. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 147–163.

[20] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K Marina. 2017. Network slicing in 5G: Survey and challenges. *IEEE communications magazine* 55, 5 (2017), 94–100.

[21] Saurabh Ganeriwal, Ram Kumar, and Mani B Srivastava. 2003. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems.* 138–149.

[22] Francisco M García, Rubén Moraleda, Santiago Schez-Sobrino, Dorothy N Monekosso, David Vallejo, and Carlos Glez-Morcillo. 2023. Health-5g: a mixed reality-based system for remote medical assistance in emergency situations. *IEEE Access* 11 (2023), 59016–59032.

[23] Alex Graves and Alex Graves. 2012. Long short-term memory. *Supervised sequence labelling with recurrent neural networks* (2012), 37–45.

[24] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2016), 2222–2232.

[25] Saikat Guha, Prithwish Basu Basu, Chi-Kin Chau Chau, and Richard Gibbens. 2011. Green wave sleep scheduling: optimizing latency and throughput in duty cycling wireless networks. *IEEE Journal on Selected Areas in Communications* 29, 8 (2011), 1595–1604.

[26] Benjamin R Hamilton, Xiaoli Ma, Qi Zhao, and Jun Xu. 2008. ACES: Adaptive clock estimation and synchronization using Kalman filtering. In *Proceedings of the 14th ACM international conference on Mobile computing and networking.* 152–162.

[27] Yitao Han, Liang Liu, Lingjie Duan, and Rui Zhang. 2020. Towards reliable UAV swarm communication in D2D-enhanced cellular networks. *IEEE Transactions on Wireless Communications* 20, 3 (2020), 1567–1581.

[28] Rudolph Emil Kalman. 1960. A new approach to linear filtering and prediction problems. (1960).

[29] Hayang Kim, Xiaoli Ma, and Benjamin Russell Hamilton. 2011. Tracking low-precision clocks with time-varying drifts using Kalman filtering. *IEEE/ACM Transactions on Networking* 20, 1 (2011), 257–270.

[30] Kyeong Soo Kim, Sanghyuk Lee, and Eng Gee Lim. 2016. Energy-efficient time synchronization based on asynchronous source clock frequency recovery and reverse two-way message exchanges in wireless sensor networks. *IEEE Transactions on Communications* 65, 1 (2016), 347–359.

[31] Joseph J LaViola Jr. 2000. A discussion of cybersickness in virtual environments. *ACM Sigchi Bulletin* 32, 1 (2000), 47–56.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 2002. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (2002), 2278–2324.

[33] Yu-Ngok Ruyue Li, Mengzhu Chen, Jun Xu, Li Tian, and Kaibin Huang. 2020. Power saving techniques for 5G and beyond. *IEEE Access* 8 (2020), 108675–108690.

[34] Debby Lin, Gilles Charbit, and I-Kang Fu. 2015. Uplink contention based multiple access for 5G cellular IoT. In *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall).* IEEE, 1–5.

[35] Simon Lynen, Markus W Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. 2013. A robust and modular multi-sensor fusion approach applied to MAV navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems.* IEEE, 3923–3929.

[36] James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Vol. 5. University of California press, 281–298.

[37] Sathiya Kumaran Mani, Ramakrishnan Durairajan, Paul Barford, and Joel Sommers. 2016. MNTP: Enhancing time synchronization for mobile devices. In *Proceedings of the 2016 Internet Measurement Conference.* 335–348.

[38] Miklós Maróti, Branislav Kusy, Gyula Simon, and Akos Lédeczi. 2004. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems.* 39–49.

[39] David L Mills. 1989. On the accuracy and stablility of clocks synchronized by the network time protocol in the internet system. *ACM SIGCOMM Computer Communication Review* 20, 1 (1989), 65–75.

[40] David L Mills. 2002. Internet time synchronization: the network time protocol. *IEEE Transactions on communications* 39, 10 (2002), 1482–1493.

[41] David L Mills. 2006. Simple Network Time Protocol (SNTP).

[42] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. 2021. *Introduction to linear regression analysis.* John Wiley & Sons.

[43] Nordic Semiconductor. 2025. nRF9160 DK Development Kit. https://www.nordicsemi.com/Products/Development-hardware/nRF9160-DK. Accessed July 2, 2025.

[44] Andrew N Novick and Michael A Lombardi. 2015. Practical limitations of NTP time transfer. In *2015 joint conference of the IEEE international frequency control symposium & the European frequency and time forum.* IEEE, 570–574.

[45] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and JD1030 Tygar. 2001. SPINS: Security protocols for sensor networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking.* 189–199.

[46] Ceferino Gabriel Ramirez, Anton Sergeyev, Assya Dyussenova, and Bob Iannucci. 2019. LongShoT: Long-range synchronization of time. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks.* 289–300.

[47] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. 2017. Low power wide area networks: An overview. *ieee communications surveys & tutorials* 19, 2

(2017), 855–873.
[48] Lisa Rebenitsch and Charles Owen. 2016. Review on cybersickness in applications and visual displays. *Virtual reality* 20 (2016), 101–125.
[49] Victor S Reinhardt. 2005. A review of time jitter and digital systems. In *Proceedings of the 2005 IEEE International Frequency Control Symposium and Exposition, 2005.* IEEE, 38–45.
[50] Zenepe Satka, Mohammad Ashjaei, Hossein Fotouhi, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. 2023. A comprehensive systematic review of integration of time sensitive networking and 5G communication. *Journal of systems architecture* 138 (2023), 102852.
[51] Thomas Schmid, Zainul Charbiwala, Jonathan Friedman, Young H Cho, and Mani B Srivastava. 2008. Exploiting manufacturing variations for compensating environment-induced clock drift in time synchronization. *ACM SIGMETRICS Performance Evaluation Review* 36, 1 (2008), 97–108.
[52] Thomas Schmid, Zainul Charbiwala, Roy Shea, and Mani B Srivastava. 2009. Temperature compensated time synchronization. *IEEE Embedded Systems Letters* 1, 2 (2009), 37–41.
[53] Ashish Kumar Sultania, Pouria Zand, Chris Blondia, and Jeroen Famaey. 2018. Energy Modeling and Evaluation of NB-IoT with PSM and eDRX. In *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 1–7.
[54] Zhaowei Tan, Jinghao Zhao, Yuanjie Li, Yifei Xu, and Songwu Lu. 2021. {Device-Based}{LTE} latency reduction at the application layer. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 471–486.
[55] Biruk E Tegicho, Tadilo E Bogale, Abdullah Eroglu, Zhijian Xie, and William Edmonson. 2023. Intra-UAV Swarm Connectivity in Unstable Environment. *IEEE Transactions on Vehicular Technology* 72, 11 (2023), 13929–13939.
[56] Robert R Tenney and Nils R Sandell. 2007. Detection with distributed sensors. *IEEE Transactions on Aerospace and Electronic systems* 4 (2007), 501–510.
[57] TensorFlow Team. 2021. TensorFlow Lite Micro: Machine Learning for Microcontrollers. https://github.com/tensorflow/tflite-micro. Accessed: 2025-07-02.
[58] Kohei Tsurumi, Aoto Kaburaki, Koichi Adachi, Osamu Takyu, Mai Ohta, and Takeo Fujii. 2022. Simple clock drift estimation and compensation for packet-level index modulation and its implementation in LoRaWAN. *IEEE Internet of Things Journal* 9, 16 (2022), 15089–15099.
[59] Heng Wang, Yan Zou, Xiaojiang Liu, and Min Li. 2024. A fast convergence scheme for distributed consensus time synchronization using multi-hop virtual links in industrial wireless sensor networks. *IEEE Sensors Journal* (2024).
[60] Dominik Welte, Christopher Lehmann, Manuel Schappacher, Thomas Höschele, Axel Sikora, and Frank HP Fitzek. 2024. Tsn over 5g: Overcoming challenges and realizing integration. In *2024 IEEE 20th International Conference on Factory Communication Systems (WFCS)*. IEEE, 1–8.
[61] Zhuqing Xu, Junzhou Luo, Zhimeng Yin, Tian He, and Fang Dong. 2020. S-MAC: Achieving high scalability via adaptive scheduling in LPWAN. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 506–515.
[62] Ting Yang, Yuqing Niu, and Jiexiao Yu. 2020. Clock synchronization in wireless sensor networks based on Bayesian estimation. *IEEE Access* 8 (2020), 69683–69694.
[63] Hüseyin Yiğitler, Behnam Badihi, and Riku Jäntti. 2020. Overview of time synchronization for IoT deployments: Clock discipline algorithms and protocols. *Sensors* 20, 20 (2020), 5928.
[64] Ali Zaidi, Anders Bränneby, Ala Nazari, Marie Hogan, and Christian Kuhlins. 2020. *Cellular IoT in the 5G era*. Technical Report. Ericsson. https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-iot-in-the-5g-era
[65] Dimitrios Zorbas, Khaled Abdelfadeel, Panayiotis Kotzanikolaou, and Dirk Pesch. 2020. TS-LoRa: Time-slotted LoRaWAN for the industrial Internet of Things. *Computer Communications* 153 (2020), 1–10.
[66] Kingsley Jun Zou, Kristo Wenjie Yang, Mao Wang, Bingyin Ren, Jingsong Hu, Jingjing Zhang, Min Hua, and Xaohu You. 2015. Network synchronization for dense small cell networks. *IEEE Wireless Communications* 22, 2 (2015), 108–117.