# Reliable Firmware Updates for the Information-Centric Internet of Things

Cenk Gündoğan
HAW Hamburg
cenk.guendogan@haw-hamburg.de

Christian Amsüss
christian@amsuess.com

Thomas C. Schmidt
HAW Hamburg
t.schmidt@haw-hamburg.de

Matthias Wählisch
Freie Universität Berlin
m.waehlisch@fu-berlin.de

## ABSTRACT

Security in the Internet of Things (IoT) requires ways to regularly update firmware in the field. These demands ever increase with new, agile concepts such as security as code and should be considered a regular operation. Hosting massive firmware roll-outs present a crucial challenge for the constrained wireless environment. In this paper, we explore how information-centric networking can ease reliable firmware updates. We start from the recent standards developed by the IETF SUIT working group and contribute a system that allows for a timely discovery of new firmware versions by using cryptographically protected manifest files. Our design enables a cascading firmware roll-out from a gateway towards leaf nodes in a low-power multi-hop network. While a chunking mechanism prepares firmware images for typically low-sized maximum transmission units (MTUs), an early Denial-of-Service (DoS) detection prevents the distribution of tampered or malformed chunks. In experimental evaluations on a real-world IoT testbed, we demonstrate feasible strategies with adaptive bandwidth consumption and a high resilience to connectivity loss when replicating firmware images into the IoT edge.

## CCS CONCEPTS

• **Networks** → **Network protocol design**; **Network reliability**; *Network experimentation*; • **Computer systems organization** → *Embedded and cyber-physical systems.*

## KEYWORDS

Constrained IoT, ICN, firmware updates, security, performance measurement

**ACM Reference Format:**
Cenk Gündoğan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. 2021. Reliable Firmware Updates for the Information-Centric Internet of Things. In *8th ACM Conference on Information-Centric Networking (ICN '21), September 22–24, 2021, Paris, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3460417.3482974


*ICN '21, September 22–24, 2021, Paris, France*

ACM ISBN 978-1-4503-8460-5/21/09…$15.00
https://doi.org/10.1145/3460417.3482974

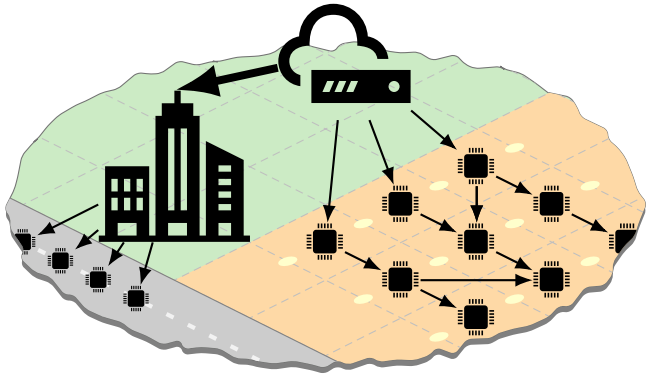

**Figure 1: Massive firmware roll-out campaign in distributed and heterogeneous networks**

## 1 INTRODUCTION

The deployment of Information-Centric Networking (ICN) [4, 51] on embedded wireless devices [38] was first considered a decade ago. With the advent of the Internet of Things (IoT), early experiments [10] confirmed benefits for constrained multi-hop networks from operating NDN [22, 53] as a network layer directly on top of data links. Since then a large body of work has proposed and evaluated ICN in the IoT context [1, 5, 7, 10, 14, 17, 40, 44, 45]. Essential findings show that hop-wise forwarding with caching and a leaner network stack improve network performance over IP as well as the adaptability to lossy regimes.

Regular software updates are part of the common life cycle for today's computer systems, and increasing security and agility demands require a similar practice for the IoT. The distribution of firmware or application updates on the Internet, however, is one of the most challenging and resource-consuming tasks [28]. Major updates from popular vendors are repeatedly visible as peak loads at Internet exchange points. Consequently, it is natural to question whether the constrained, lossy networks of the IoT can carry such burdens and how update campaigns may perform.

In this work, we devise and evaluate procedures for reliable, secure, yet scalable software updates in the constrained IoT. Our target objective is the massive roll-out of firmware in edge networks as visualized in Fig. 1. We want to prove feasibility by leveraging the potential benefits of ICN forwarding and caching. Our key contributions are (*i*) context-specific naming, version discovery,

and verification, (*ii*) scalable and reliable chunk distribution across updating network nodes with inbuilt DoS detection, (*iii*) thorough experimental evaluations of different update strategies in a real testbed with realistic multi-hop radio links.

In this paper, we can show that reliable updates of large firmwares in deep multi-hop topologies are indeed feasible. Our findings indicate that firmware dissemination in large networks nested up to seven hops complete within 10 to 30 minutes. We also observe that rapid roll-outs in these networks will fully exhaust resources, whereas slower, cascading update strategies leave sufficient resources at intermediate nodes for continued operations.

The remainder of this paper is structured as follows. We introduce the problem of firmware propagation in the IoT along with related work in Section 2. Section 3 presents the core concepts of secure and reliable firmware updates. A thorough evaluation and discussion of the results follows in Section 4. Finally, we conclude with an outlook in Section 5.

## 2 THE PROBLEM OF FIRMWARE PROPAGATION AND RELATED WORK

### 2.1 Challenges in low-power regimes

Secure firmware roll-out campaigns for large-scale IoT deployments demand a coordinated interaction with great regularity between multiple stakeholders. Vendors prepare and publish firmware versions and local site administrators oversee the roll-out procedure. An autonomous firmware update without physical proximity can drastically reduce the roll-out time and management overhead for local site administrators. IoT devices connect through low-power and lossy networks (LLNs) to powerful border routers. Especially in industrial and rural settings where infrastructure is challenged by natural and regulatory constraints, wireless multi-hop networks are prominent and continuous access to deployed hardware is not always feasible. These regimes are subject to radio interferences and individual link error probabilities that accumulate in a destructive manner. In addition, limited maximum transmission units as well as low bandwidth and high delay link capabilities further complicate the distribution of large firmware objects, which necessarily split into hundreds to thousands of fragments. While corrective actions on the link, network, and application layer usually recover packet loss, small amounts of retransmissions behave additive and induce link stress in broadcast range, which impacts energy expenditures of battery-operated devices. The exhaustive task of delivering image files also opens up significant attack vectors for denial of service (DoS) attempts. Willfully tampered or inadvertently modified firmware images deplete network and memory resources to a point where devices neglect mission-critical duties.

The importance of well-thought-out firmware roll-out architectures that efficiently operate in low-power regimes and display a resilient security posture is undisputed. Several approaches have been proposed in research or have already been deployed in industrial solutions.

### 2.2 Firmware updates in the IoT

SUIT [34] is a recent addition to the menagerie of firmware update architectures. It is driven by the eponymous IETF working group and aims for a standardized update mechanism in constrained IoT networks that is reliable and secure. SUIT specifies a concise and machine-processable manifest document [32, 33] that describes meta-data of firmware images, such as their download location, firmware version, and optional processing steps to decompress and decrypt binaries. This architecture relies on the Internet protocol stack for retrieving updates and therefore expects certain protocol mechanisms to be present, like congestion and flow control, packet fragmentation, and the ability to resume corrupted transfers. Given its current momentum at the IETF, we consider SUIT as a suitable blueprint for our information-centric firmware update approach.

ZigBee [56] is a protocol specification harboring various network solutions to inter-connect a wide range of heterogeneous, ZigBee certified devices. It builds on IEEE 802.15.4 and is prominently used by several product lines, such as Philips Hue, OSRAM lightify, and some Xiaomi devices, albeit not always securely [41]. In the ZigBee Over the Air (OTA) Upgrade Cluster module, clients regularly poll firmware information, or a server performs *Image Notify* push operations for clients not in hibernation. The distribution of upgrade images via broadcast or multicast is not recommended due to a missing point-to-point security. In this case, ZigBee advises a separate unicast attestation with the upgrade server after completing an image transfer. In contrast to ZigBee, we believe the vendor-independent manifest files of SUIT to concisely organize meta-data provide a greater accessibility to the update process in heterogeneous network deployments.

### 2.3 Reliable content transfers and data management in constrained networks

Large data objects, such as uncompressed binary images with moderate software complexity for embedded devices, can reach file sizes in the range of tens to hundreds of kilobytes. Prior to the IoT era, wireless sensor networks (WSNs) explored network reprogrammability of low-power devices in broadcast media and reliably disseminated large data objects (*e.g.,* a firmware) using epidemic routing methodologies [21, 27, 49]. Delicate adjustments to the classic flooding, such as node density awareness, windowing, the use of negative acknowledgments (NACKs), and unicast requests with broadcast data transmissions have shown promising results in lossy networks. Due to the generally inconsistent protocol layering in former WSNs, packets exceeding link MTUs in constrained network environments had to be fragmented on the application level.

In contrast, the current IoT mostly builds on IPv6 and to bypass transmission limits of typical link layers, the IETF designed 6LoWPAN [31]—a convergence protocol to adapt IPv6 functionalities to challenging LLNs. It supports a header compression to reduce header verbosity and a fragmentation scheme [31], which caps at 2048 bytes and is therefore inoperable for firmware propagations. The constrained application protocol (CoAP) [46] is part of the IETF envisioned IoT network stack and supplements IoT networks with a RESTful communication paradigm. Block-wise transfer [12] is an add-on to CoAP for splitting payload into equally sized blocks, which are then iteratively transmitted with minimal server-side state. Chunking on the CoAP level further enables the use of CoAP reliability features for each separate block.

Recent studies [6, 17, 30] reveal a superior data delivery performance for named-data networking (NDN) [22] in low-power

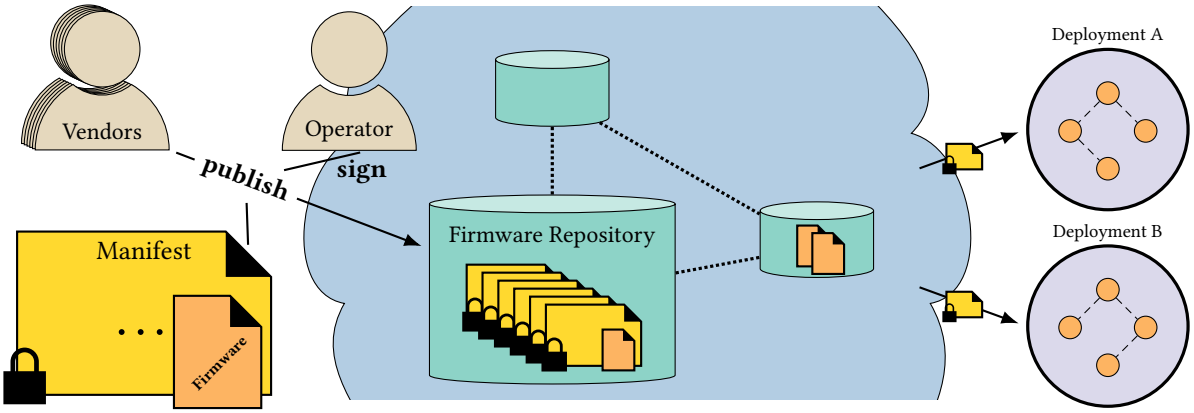


**Figure 2: Overview on the back-end system of our information-centric, reliable firmware roll-out approach.**

networks compared to end-to-end IoT protocols, such as CoAP and the message queuing telemetry transport for sensor networks (MQTT-SN) [48]. NDN leaves the fragmentation of larger named-data objects to upper layers, since naming decisions for newly created chunks are highly application-specific. Link fragmentation extensions [18, 36, 47] operate below NDN and modify the packet structure. Other approaches [13, 19, 35] apply a fragmentation and naming scheme on the application to yield a structured access to data chunks with predictable names.

## 3 BUILDING BLOCKS FOR RELIABLY UPDATING FIRMWARE WITH NDN

### 3.1 Roll-out campaign management

We design a secure and reliable campaign management system for firmware roll-outs that handles the delivery of software updates to numerous constrained edge devices in multiple sites using NDN. We use the SUIT [34] model as a blueprint for our information-centric approach and adopt essential system components and the same terminology. Figure 2 illustrates our name-based back-end proposal, which consists of three components: (*i*) publishing and versioning firmware images and manifest files by vendors, (*ii*) managing the storage of chunked software updates by an operator and providing access to the IoT deployment sites, and (*iii*) a timely notification of version updates and a reliable delivery of necessary updates towards edge devices on the IoT side.

### 3.2 Firmware preparation and publication

**Namespace management.** Large site deployments can consist of heterogeneous devices from varying vendors and the highest level of interoperability is essential to construct an energy-efficient system. A systematic namespace management regulates all interactions between vendors and IoT devices. Figure 3 demonstrates our name schema used for all components, ranging from upper-layer application functions down to forwarding and caching duties.

Each deployment has a globally unique name and may identify an offshore drilling rig, segments of a connected urban network, or a smart home environment. We consider the deployment identifier as the leading component in our name schema to keep forwarding states towards single deployment sites minimal, *i.e.,* they most certainly aggregate due to the spatial proximity of devices within the scope of a deployment. Vendor names are equally globally unique like deployment identifiers and both components are managed by the same, external registry. Finally, a device class designates a specific firmware for all nodes of the same product type. The timestamp component describes the actuality of a firmware and is encoded as a Unix timestamp with a predefined granularity. To fully leverage the in-network caching abilities of NDN, binaries are prepared for device classes instead of yielding unique binaries for each single device. This also reduces the binary management overhead on the vendor site.

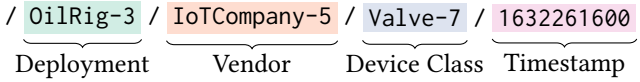


**Figure 3: Namespace schema.**

**Firmware generation.** Vendors precompile firmware images for their deployed product lines and keep track of the software versioning. Since binaries are prepared for device classes, the images cannot ship with sensitive data. Device-specific configurations are rather obtained on run-time after a successful firmware installation on an IoT node. This requires that each IoT device is provisioned with vendor-specific data for bootstrapping purposes during the manufacturing stage or with the use of an out-of-band channel, which is already common practice for real-word deployments. This data outlasts firmware upgrades and is stored independently of the program code, *e.g.,* in a dedicated address space on the flash memory, or using an SD card. To protect the firmware integrity, vendors also generate a message digest of the binary alongside the firmware image.

**Preparation of firmware chunks.** The small-sized MTUs in common network link technologies disallows the transmission of images in single network packets. For a successful delivery, an image fragmentation at the vendor and a reassembly at the IoT edge devices is necessary. Fragmentation on convergence layers [18, 31] is a solution to provide a hop-wise, fragmented delivery between

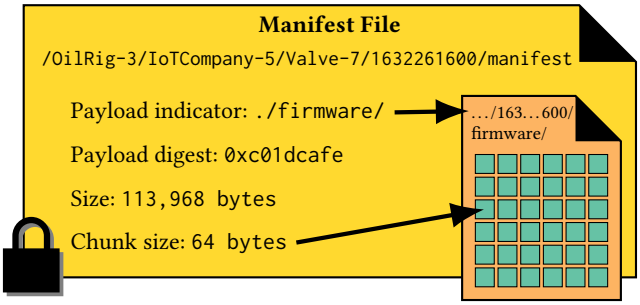


**Figure 4: Manifest description and fixed-length chunks of a corresponding firmware image.**

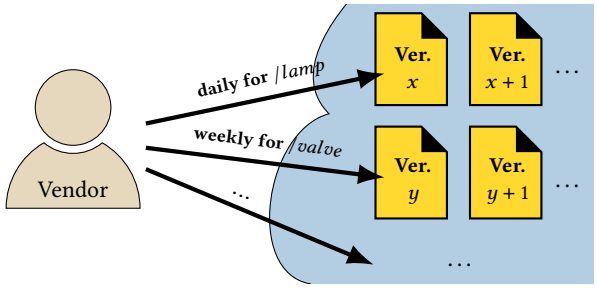


**Figure 5: Vendor publishes firmwares and aligns date granularity with polling interval of device classes.**

two peers, but due to the layering, these schemes make the caching of individual fragments impossible. Thus, we focus on a fragmentation approach that chunks the image on application level and reassembles them at the IoT edge device after all chunks have been successfully retrieved.

The reassembly of fragmented images must be as simplistic as possible for the constrained devices. We therefore follow a linear chunking of the image file in our solution, where each chunk is of fixed length (the last chunk being an exception). The reconstruction on the low-power devices is straightforward as fixed-length chunks can be joined using offsets, which makes the need for an ordered delivery unnecessary. Chunk sizes may vary between device classes, since different link-layers will yield different MTUs. Each chunk is addressed by appending a monotonically increasing chunk identifier */chunk/id* to the base name (see Figure 3), starting at */chunk/0*.

**Manifest description.** As demonstrated in Figure 4, a vendor also creates a manifest file following the SUIT model to organize meta-data on the firmware version and binary image. They include the binary size and message digest as well as parameters for the chunking algorithm. To preserve the authenticity of manifest files, vendors sign them during the upload process. This also protects the message digest, which is later used to validate the final firmware image on the IoT devices. The manifest is addressed using the base name (see Figure 3) and the suffix */manifest*.

**Firmware upload and binary management.** Once all artifacts have been produced, a vendor delivers the manifest and firmware chunks to the corresponding deployment operator to serve them in a firmware repository. The publication process runs in an automated manner and requires an authentication framework to ensure consistency and security, such as the publicly auditable bookkeeping service NDN DeLorean [52]. A firmware repository stores versioned images of all vendors and retains them until they are purged. For replication purposes, an operator can deploy multiple firmware repository instances, which then synchronize using any data set synchronization solution [54, 55].

The uploaded firmware binaries and manifests are tagged following the naming scheme in Figure 3. The suffix for the actual image is */firmware* and the corresponding manifest is */manifest*; chunks are accessed via */chunk/id*. The timestamp in the naming scheme updates for new firmware versions to reflect the upload time and the granularity of the epoch time is coordinated with the polling interval of the devices, *e.g.,* a daily alignment on midnight would yield `1632261600` for `09/22/2021 00:00:00`. A vendor chooses different degrees of granularity on a device class level as illustrated in Figure 5.

**Discussion: full versus incremental updates.** We design our firmware roll-out approach to always deliver the full binary. An alternate approach would explore the use of differential algorithms to compute software differences, *e.g.,* with *bsdiff* [39], and transmit them in the form of binary patches. While it is undemanding for powerful firmware repositories to calculate a minimal diff representation, the patch size can grow very quickly for compiled binaries. Especially in the IoT, binaries are compiled with optimizations to reduce the binary size as far as possible to fit the image on the programmable flash memory. This can lead to large differences for small changes between software versions due to code re-organizations, up to the point, where caching them in the content store becomes unfeasible and would evict application data.

Incremental updates using the linear chunking approach is another alternative, which appears to be attractive on first sight. Chunks from previous versions could be reused with a correct name mapping in the manifest file to reorder the fragments independent of their sequential chunk identifier. However, this can quickly inflate the size of the meta-data itself and may require a separate fragmentation for the manifest file.

Sophisticated linking techniques that use auxiliary information about the structure of deployed software modules [25] can produce concise patches compared to naïve diff algorithms which operate on the byte level. Run-time relinking of software components directly on the sensor nodes can lead to minimal diff representations, but requires an extensive tooling support during binary compilation and software installation [29]. While we only focus on the propagation of the binary, the actual representation of such an artifact (full binary or an increment) is rather secondary and may only necessitate slight protocol adaptations regarding the naming schema.

## 3.3 Firmware update process

**Firmware version discovery.** IoT devices are naturally provisioned with an up-to-date firmware version before they become operational in a deployment. Over time, vendors release new software versions to update device functionalities or to handle security related issues. Depending on the network availability, a device may

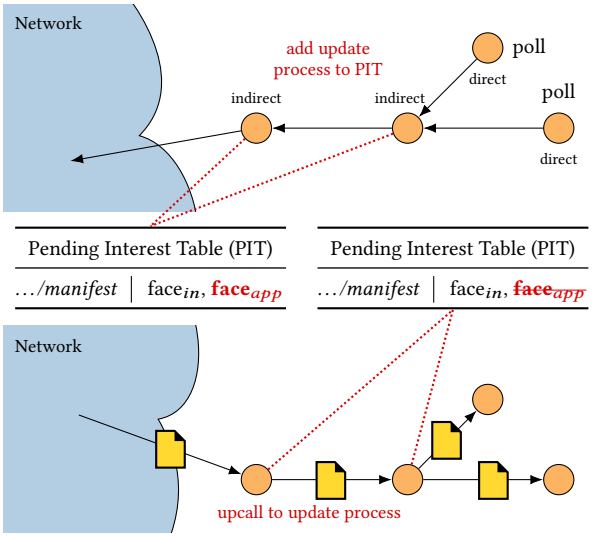


**Figure 6: Direct and implicit version discovery.**

be a single or multiple versions behind the current firmware. A version discovery is therefore the first step to any upgrade process.

Two fundamental strategies exist when determining the availability of a new firmware update: (*i*) proactively notifying the IoT devices using push mechanics, and (*ii*) periodically polling the firmware repository. While timely notifications from a firmware server to the IoT device minimize the operational run time of outdated software components, it also bears the following issues. First, notifications are not guaranteed to arrive in low-power regimes where nodes favor extended sleep cycles. Second, it requires server-side state and maintenance overhead to keep track of deployed versions as well as topological information to ensure node reachability, and last, the push mechanism is not native to NDN.

Our approach primarily relies on a pull-driven version discovery, where the embedded devices periodically request the latest manifest file for a dedicated time frame. Vendors convey a sensible polling interval on device class level, *e.g.,* a daily check on midnight for remotely deployed gas valves, or flexible intervals based on harvested energy levels for battery-less sensors. These guidelines are programmed during run-time configurations and may change at an operator's discretion. Since the Unix epoch denotes the actuality of a firmware image in the name schema, all IoT devices need to re-adjust drifting system clocks using an external mechanism, *e.g.,* by relying on the time information of an equipped GPS module, or by operating a time synchronization protocol, such as NDNTP [37].

**Retrieval of firmware versions.** To discover a new version, IoT devices send Interests to the name that identifies the latest firmware version by setting the correct time frame. Following our previous example, the Interest may describe the name */OilRig-3/IoTCompany-5/Valve-7/1632261600/manifest*, in which the requested time frame is greater than the time frame of the locally running firmware. The firmware repository returns a manifest file if the requested update is available, *i.e.,* a vendor published the binary image for the specified time frame. Interest retransmissions retry the update request for a configurable, but limited number of times to recover manifests from packet loss. In the event that a requested manifest does not exist yet or all corrective actions fail, the Interest times out as part of the default NDN forwarding logic and the IoT node triggers a subsequent update request on the next polling interval, potentially on midnight of the next day. Negative acknowledgments for Interests (NACKs) is a supported NDN protocol element to hint at the absence of requested data or to carry nuanced error codes of the application. For our retrieval mechanism, they may include application-level indications about the latest firmware version. While NACKs are not necessary to ensure a continuous operation, this feature (*i*) provides an optimization to reduce the amount of retransmissions when polling for a new, non-existent firmware version, and (*ii*) assists with the convergence of updates for devices that missed a version publication, *e.g.,* due to network unavailability. The lifetimes of cacheable NACKs need to be aligned with the firmware release cycles to prevent them from wrongly satisfying requests for eventually released versions. To reduce the attack surface, NACKs require similar security considerations as manifest packets.

IoT nodes may be disconnected for longer periods from the core network and thereby may fall several versions behind. Fixing a maximal update frequency at the application level allows a node to always request the latest version at the appropriate Unix epoch. Hence, outdated devices need not attempt to retrieve obsolete versions. Forwarding states are handled by an external routing system, *e.g.,* [20, 43], preferably using a single default route from all IoT devices toward the firmware repository.

**Implicit consumption of firmware versions.** Polling intervals of devices within the same class can drift apart over time, so we utilize an implicit version discovery process to reduce the amount of individual manifest requests and to increase the reactivity of the firmware roll-out. Each IoT forwarder in a multi-hop request path compares incoming manifest requests with its own device class. On a positive match and if the requested name has a greater epoch time than the currently operating firmware, then the update process of a forwarding device internally registers to the same entry in the Pending Interest Table (PIT) as illustrated in Figure 6. This assures that each device of the same class on a request path consumes the manifest and then initiates the retrieval procedure of the firmware image before its local request interval triggers.

**Retrieval of firmware image chunks.** Once an edge node determines the need for a version upgrade by receiving an up-to-date manifest file, it prepares for retrieving the associated binary image. Initially, the manifest signature is validated using key materials previously provisioned by a vendor. On a failed check, the upgrade process aborts and this incident is reported to the vendor. A valid signature triggers the retrieval of all firmware chunks as designated by the manifest. Each chunk is addressed by appending the chunk identifier (*/chunk/id*) to the base name, where *id* starts at 0 and gradually increments to the maximum chunk number as appointed by the manifest. This also ensures that a few resources are available for alternative forwarding duties. Since memory and network resources are generally limited in low-power regimes, the system uses a stop-and-wait automatic repeat-request (ARQ) error-control method, *i.e.,* each chunk is retrieved iteratively as illustrated in Figure 7. Characteristically, resource-constrained class 2 devices [11] equip less than 100 KiB of main memory, where larger parts are

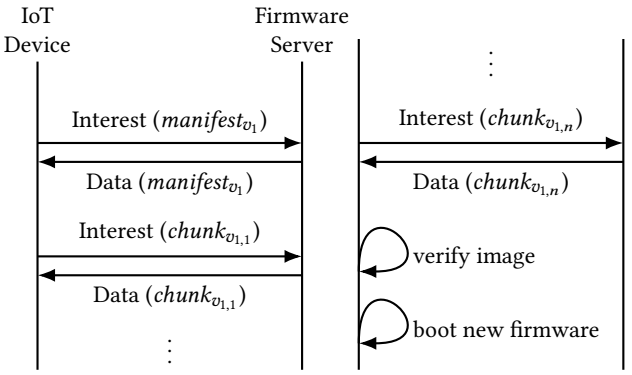


**Figure 7: Iterative retrieval of firmware chunks.**

inevitably consumed by the operating system, the network stack, and reserved for application purposes. This leaves only persistent memory components, *e.g.,* flash and SD cards, to buffer intermediate chunks during the retrieval. In contrast to the available RAM, external memory often displays storage capacities that are orders of magnitude larger, but uncoordinated access can also consume the available energy budget as I/O operations tend to be slower and energy-draining.

We define two different chunk retrieval strategies that we assess in our experimental evaluations. The first method allows concurrent firmware updates from nodes on the same request path. The second retrieval method disallows overlapping updates and rather prefers an ordered update that cascades downstream into the IoT network.

**Concurrent firmware updates.** While nodes request one chunk at a time, they still perform forwarding duties for other devices. Overlapping upgrade processes may also yield incoming data objects that are farther advanced in the firmware buffer than the local chunk identifier as illustrated in Figure 8. In this case, a firmware consumer diverts matching chunks with higher progression into the local buffer. Simultaneously, this buffer is also used for serving incoming chunk requests from other devices. Although this optimization results in an unordered data retrieval, the use of fixed-length chunks eliminates the need for reorganizing the fragments when reconstructing the image. Power demanding I/O operations to persistent memory are thus minimized.

**Cascading firmware updates.** In this retrieval method, a node denies the delivery of firmware chunks for the same device class as long as a node did not complete the update process itself. Downstream nodes run into request timeouts for the first chunk and application retransmissions retry the retrieval using a configurable polling interval. With this strategy, firmware versions propagate hop-wise from a gateway device towards any leaf node of a multi-hop network.

**Firmware verification.** After completing the retrieval, all necessary chunks reside in the local chunk buffer and this also concludes the full image reassembly. A node calculates a message digest across the buffer and validates it against the previously received firmware digest. On a positive verification, the binary is copied to the correct flash region, the temporary chunk buffer is cleared, and the bootloader is notified to invoke the new firmware. The timer for

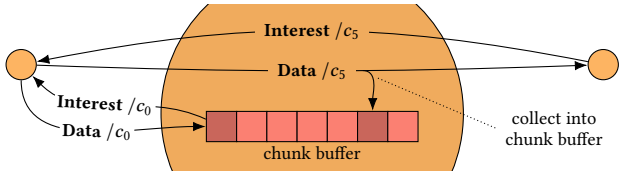


**Figure 8: Local buffer collects chunks from overlapping upgrade processes.**

the next version request is armed as soon as the new image boots successfully. Following the SUIT [34] philosophy, the update process still keeps the old firmware image on the device as a backup in case the recent firmware update breaks the node operation. At worst, the bootloader initiates a fail-safe to re-flash the old binary and return to a correct and consistent behavior.

**Firmware replication on connectivity loss.** Once the upgrade completes, a device can also serve the latest manifest and binary chunks to downstream devices. The advantage of using a linear binary chunking is that an up-to-date forwarder device serves chunk requests directly from its read-only flash region where the currently running firmware resides, without separately consuming main memory. A firmware version can therefore cascade downstream into the IoT network in a hop-by-hop fashion without necessary operations from the firmware server. This design confines chunk retrievals to a single link and therefore leads to a reduction in bandwidth usage. It also provides a loose coupling, so that upgrade processes become resilient to uplink outages and are unaffected by temporary network disruptions.

**Early denial of service (DoS) detection.** Images may consist of hundreds or thousands of chunks, depending on the firmware complexity and the (usually small) MTUs of underlying link-layer technologies. NDN protects singular content objects (see Figure 9a), but (*i*) the chunk-wise computation of digital signatures using asymmetric cryptography is infeasible for the constrained environment, in particular if no hardware acceleration is available [24]; (*ii*) full-length signatures inflate each packet, thereby immensely reducing the actual goodput of the firmware delivery, and (*iii*) IoT devices must store message signatures alongside the respective data to serve requests from the local cache. This consumes a storage capacity that can grow as large as the firmware itself in low MTU scenarios (*e.g.,* for 802.15.4 with less than 128 bytes payload room).

Figure 10 illustrates the aggravating effect of comparatively large signature sizes. In this example, we assume the 802.15.4 MTU, a data name of 16 bytes, a structural NDN encoding overhead of another 16 bytes, and the link-layer header further consumes 23 bytes when using the long MAC address mode. This sums up to 55 bytes and leaves 73 bytes for the payload and signature. The Edwards-Curve Digital Signature Algorithm (EdDSA) [23] is a prominent choice in the IoT as it provides a high performance and relatively small signatures of 64 bytes—at least with the Ed25519 curve. Yet this reduces the available space for application data down to 9 bytes, resulting in numerous chunk packets containing individual signatures. Even for small firmware sizes of 36 KiB, 4000 chunk transmissions accumulate to a signature overhead of 256 KiB. For larger images, this linearly increases: a firmware with 144 KiB requires 16000 chunk

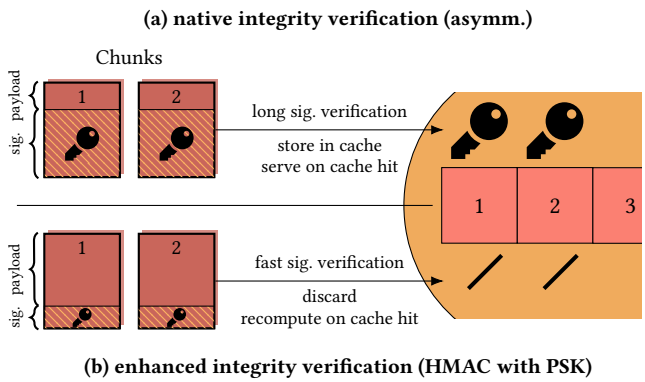


**Figure 9: Enhanced chunk-wise integrity verification to save device and network resources compared a native NDN protection with asymmetric cryptography.**

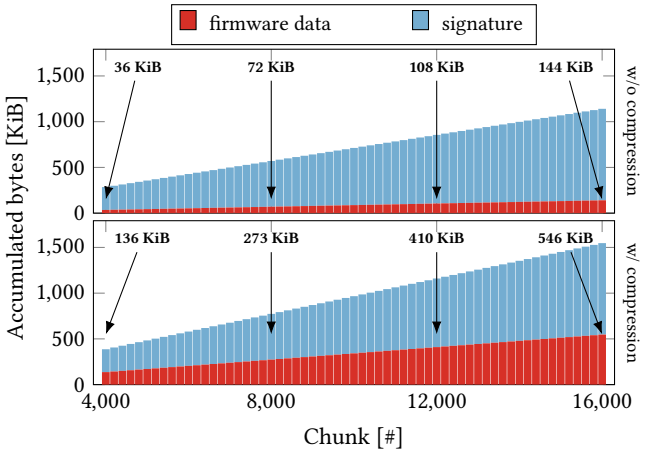


**Figure 10: Chunk-wise signature overhead compared to the actual firmware data. Chunks contain 9 bytes (w/o ICNLoWPAN compression) and 35 bytes (w/ ICNLoWPAN compression) of application data. Signatures are 64 bytes for EdDSA (Curve25519).**

transmissions and produce a signature overhead of 1 MiB. The ICNLoWPAN convergence layer [18] can remove names from Data messages and reduces the structural header overhead. Following our exercise, these enhancements increase the available space for firmware data from 9 to 35 bytes, thereby requiring four times less chunks to complete the firmware delivery. Regardless, the signature overhead remains intolerable. The severity shows in NDN cache environments where each signature has to be stored alongside the chunk data due to the asymmetric aspect of this signature algorithm that prevents IoT devices from generating them.

The integrity and authenticity of a firmware image is validated against the protected message digest from the corresponding manifest file once all chunks have been received and reassembled. Hence, signatures of individual NDN messages are redundant and we omit for the sake of efficiency. Unauthenticated packets, though, open a forceful attack vector to exhaust the resources of the IoT network: Injecting (even few) illegitimate chunks violates the integrity of the firmware and an identification of these invalid chunks is difficult after firmware reassembly. The only approach to recover the binary is then to repeatedly request the firmware, which requires all chunks to traverse the network first. To save device and network resources, a detection of erroneous deliveries and an early exit of the retrieval process is desired.

We augment individual chunks with a keyed-hash message authentication code (HMAC [26]) that is verified upon reception (see Figure 9b). Next to the asymmetric cryptography, NDN already provides the protocol elements to encode a 32-byte HMAC authentication code. To check for data integrity as well as authenticity, the HMAC requires seeding. For this early DoS detection module, we assume a pre-shared secret at all devices of a class, which can be pre-installed by the vendor or obtained in an out-of-band manner and eventually protected in secure memory. A chunk is then recorded in the chunk buffer only after correctly verified by its recipient. If a chunk validation fails, a recipient repeats requests for invalid chunks only. After iterated (*e.g.,* three) failing verification attempts, a node marks the firmware as irrecoverable, aborts the update process, and notifies the vendor.

It is noteworthy that these signature hashes based on pre-shared secrets can be discarded during caching in the chunk buffer, since nodes of the same device class can easily re-generate them using the same secret at any time. This relieves storage capacities, while preserving an intact cache operation for incoming chunk requests. For low-power regimes with small-sized MTUs, a full HMAC signature may occupy too many bytes in a frame. For optimization, we only transmit a configurable prefix of the hash, *e.g.,* 8, or 16 bytes. This trade-off increases the susceptibility to hash collisions, but drastically increases the goodput. Security and robustness of the final image verification remain unaffected by these optimizations.

## 4 EXPERIMENTAL EVALUATION

In this section, we quantitatively assess our previously outlined information-centric firmware update approach using a real protocol implementation and constrained nodes in a testbed.

### 4.1 Experiment setup

**Scenario and network topology.** We conduct our experiments in a wirelessly connected IoT deployment where a gateway node is situated at the network edge to provide an uplink connectivity to a set of 30 IoT devices. A new binary version is rolled out into the stub network. On system initialization, the constrained nodes statically arrange in a destination-oriented, directed and acyclic graph (DODAG) as depicted in Figure 11. DODAG topologies provide shortest paths from IoT devices to root nodes (*i.e.,* gateway or cloud) and therefore incur a minimal routing overhead for the prevalent *converge cast* scenario, *i.e.,* a large amount of traffic is directed to or from a central point. In fact, RPL [50]—the predominant routing protocol for the IoT—uses DODAGs as a fundamental part of its routing system. While we rely on a static topology in our test environment to sidestep the delays of routing convergence and to solely focus on the propagation of large data objects, an

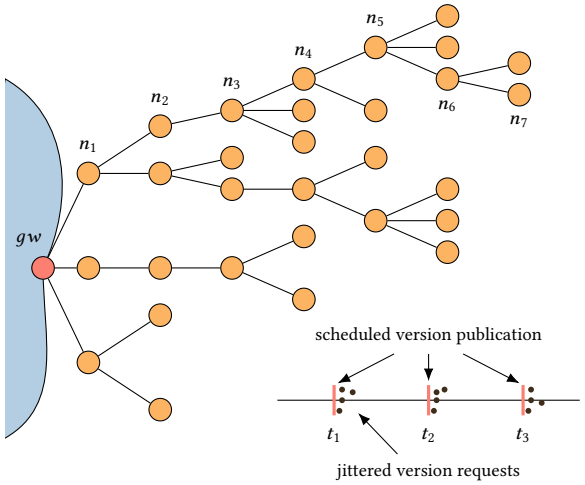


**Figure 11: Logical testbed topology modeling multiple branches from rank zero to seven of the forwarding hierarchy.**

authentic deployment would use an orthogonal routing protocol to dynamically construct and repair the DODAG as necessary.

**Software and hardware platform.** On all IoT nodes, we deploy the RIOT [9] operating system in version 2021.04. It integrates with CCN-lite, which implements a minimal NDN forwarder. The necessary update logic runs as a small IoT application using the portability layers of RIOT and CCN-lite, which opens the implementation to a wide range of hardware platforms.

We conduct all evaluations on the FIT IoT-LAB [3] testbed. It features large deployments of several ARM Cortex-M3 based class 2 devices [11] with 64 kB of RAM and 256 kB of ROM. The testbed nodes are equipped with an Atmel AT86RF231 [8] transceiver to operate on the IEEE 802.15.4 2.4 GHz radio.

**Deployment parameters.** We externally align the system clock of the IoT devices and the gateway node with the Unix epoch using the instrumentation tools of the testbed. In a configured interval of one hour, we generate new binary versions and record the corresponding manifest and image files in the content store of the gateway. Once the time is synchronized, the IoT nodes request new manifest files from the gateway node as soon as they are generated. In our experiment, we deploy the same device class throughout the network, *i.e.,* the same firmware image for all devices, but also provide a glance at the end of the evaluation on the performance for the other extreme: all nodes are of a different device class. We separately explore the two retrieval strategies: *concurrent*, where update processes overlap between multiple nodes, and *cascading*, where downstream nodes first wait for upstream nodes to complete the update.

We choose names for manifests and chunks (see Figure 3) with a total size of 45 bytes when encoded in the NDN TLV format. While we increase the image size from 32 to 128 kB in our experimental evaluations, we gradually raise the number of maximum chunks from 1000 to 4000. Thereby, we keep the chunk size fixed to 32 bytes across all configurations. This yields a length of 92 bytes for chunk data packets and the total frame size sums up to 115 bytes including the IEEE 802.15.4 link header. Thus, these parameters produce chunk packets that are very close to the link MTU of 128 bytes. The NDN forwarder performs three retransmissions in a two-second interval and the application triggers retransmissions in a jittered interval of 10±5 seconds after a designated chunk request times out. We configure three link-layer retransmissions that operate in the lower millisecond range, whereby each retransmission is slightly delayed by a random exponential backoff algorithm.

## 4.2 Firmware update progress

In our first evaluation, we gauge the update progression over time for a set of selected nodes with increasing firmware size. This nodal time measurement starts when the first firmware chunk is requested and terminates on the successful delivery of the last chunk. Our node selection consists of $n_{1...7}$, *i.e.,* the nodes that reside on the longest path in our topology. Figure 12 summarizes the various evolutions over the experiment duration.

We observe that both retrieval strategies yield very different progression charts. In the *concurrent* mode, all update procedures of $n_{1...7}$ start almost simultaneously and run concurrently for a designated time. The first two nodes $n_{1,2}$ advance with a similar chunk retrieval speed in all configurations and the remaining nodes $n_{3...7}$ display a similar alignment, albeit with a much slower evolution. While the firmware distribution with 1000 chunks continues for ≈8 minutes to complete for the whole network branch, the duration multiplies to ≈30 minutes for an image file of 128 kBytes (4000 chunks). The *cascading* deployments display the anticipated stop-and-wait characteristic. Single nodes wait for the immediate uplink node to finish the update process, before any chunk retrievals are invoked. This serialization positively affects individual update speeds. In the extreme configuration, the update duration for $n_7$ declines from 30 minutes down to 3 minutes, which is the quickest update completion on the request path. However, while individual updates appear to be faster in the *cascading* mode, the global roll-out on this path is ≈8 minutes slower than in the *concurrent* mode.

## 4.3 Goodput analysis

In our next comparison, we emphasize on nodal chunk retrieval rates to elucidate the previous progression differences. Figure 13 displays the amount of accumulated chunks that nodes retrieve in a second. We observe highly fluctuating rates throughout the update process ranging from zero chunks per second up to accumulated retrievals around 60 chunks per second. With the *concurrent* retrieval strategy, nodes $n_{3...7}$ generally display lower rates while $n_{1,2}$ have ongoing transmissions. The average performance of the $n_7$ leaf node nets to an average of approximately 2 chunks per second for all configurations. Roughly at the middle of the experiment duration the first two nodes complete their update process, which leads to slightly increased retrieval rates for the remaining nodes. This is an indication that nodes in this deployment are competing for bandwidth in the shared wireless medium. When retrievals are *cascading*, then the number of simultaneously competing nodes in the topology is drastically reduced to single nodes in all request paths of the topology that have overlapping broadcast ranges. The

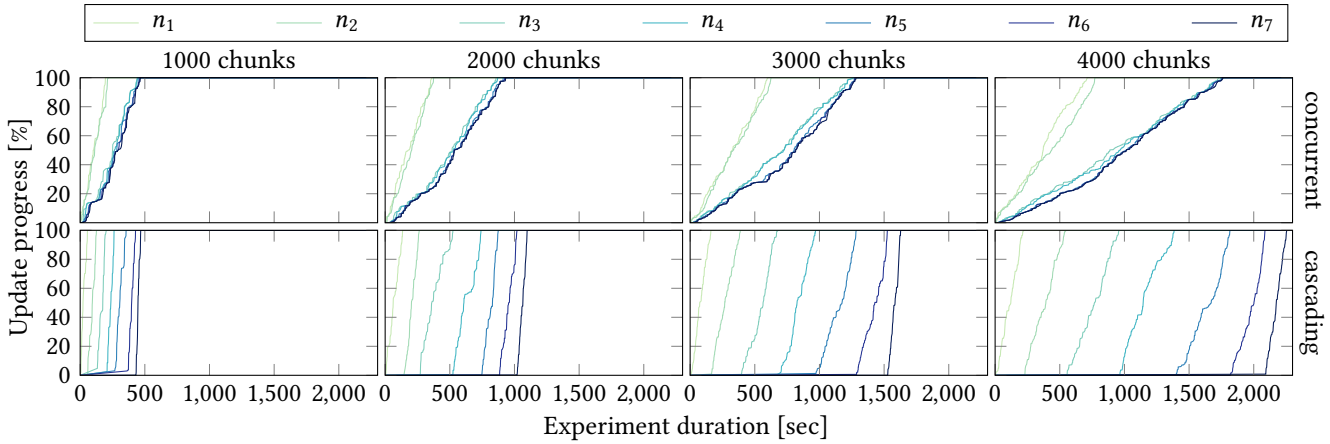


**Figure 12: Overall firmware update progression for the selected nodes $n_{1...7}$ with an increasing number of maximum chunks using the *concurrent* and *cascading* retrieval strategies.**

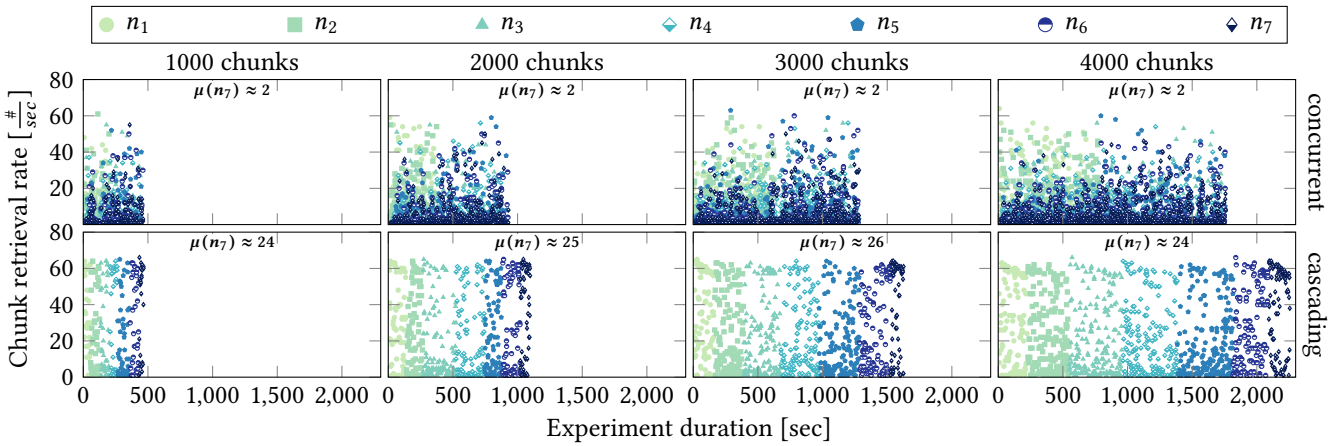


**Figure 13: Chunk retrieval rate per second for our node selection using both retrieval strategies.**

nodal goodput moderately improves for all nodes across all presented configurations. For $n_7$, this translates into a performance gain that is nearly twelve-fold. The evident oscillations are a result of request retransmissions. Unlike layer 2 retransmissions which operate on the millisecond range and are mostly invisible in the considered timescale, corrective actions on upper layers block the retrieval process by multiple seconds until messages are recovered by the network layer or application, thereby impairing the nodal goodput rates.

## 4.4 Link stress

The preceding evaluations suggest that both retrieval methods experience varying degrees of network stress when firmware updates are progressing in the multi-hop topology. We now measure the link stress for $n_7$ by quantifying the amount of retransmitted chunk requests. Figure 14 accumulates request retransmissions for blocks of 100 chunks and differentiates between corrective actions on the network and application layer. In the *concurrent* configuration, $n_7$ triggers a seemingly continuous stream of ≈5–45 retransmissions which is higher at the beginning and then slightly decreases over the experiment duration. This is in accordance with our former observation that chunk rates increase as soon as competing upstream nodes complete their updates and access to the shared medium lessens. Overall, the amount of application retransmits is rather minuscule compared to the number of network retransmissions, *i.e.,* NDN is able to recover most of the chunks with its three request attempts.

The *cascading* setup shows a much less pronounced retransmission behavior: many chunks experience no packet loss at all while other groups register less than ten network retransmissions for 100 chunks—still considerably less than the *concurrent* configuration. This relaxed progression also confirms the previously observed performance gains when firmware images are distributed in a hop-wise fashion. Application retransmissions are virtually absent, excluding the very first chunk. $n_7$ retries the retrieval of the first chunk, but $n_6$ denies the delivery until it completes its own update. This

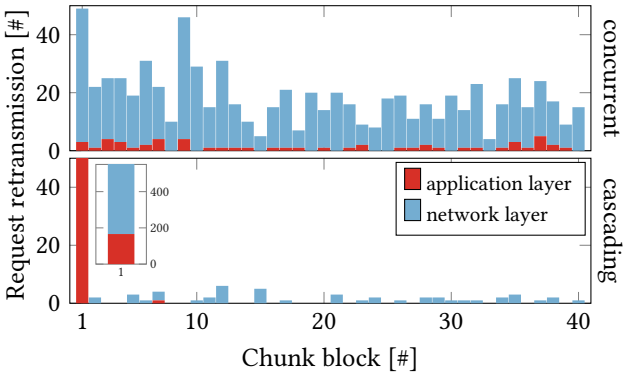


**Figure 14: Chunk request retransmissions on the application and network layer grouped into blocks of 100 chunks for the $n_7$ node.**

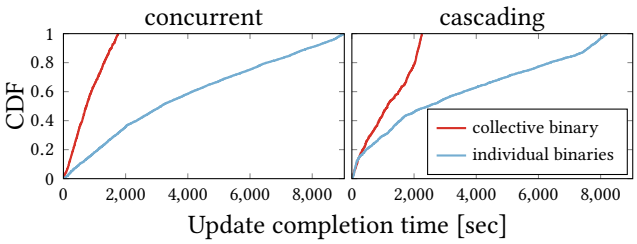


**Figure 15: Update completion time for the selected path $n_{1...7}$ with a maximum amount of 4000 chunks per firmware image.**

leads to the large amount of ≈160 application and ≈500 network retransmissions that originate from $n_7$. These numbers appear to be disproportionately high, however, these packets trigger moderately in the seconds range over a period of ≈30 minutes and do not pose a significant stress to the shared medium. Overall, we observe sufficient idle resources to continue regular network operations during the update.

### 4.5 Multiparty assessment

Up until now, our experiments updates the same device class throughout the network. A roll-out of a collective firmware image clearly benefits from the NDN multicast support: in-network caches and request aggregations can greatly balance the network utilization. In this last assessment, we configure a different device class for each device in the deployment to deliver individual binaries to the respective nodes. While this contrary extreme is usually impracticable in real-world deployments, it gives a sensible estimation on the performance of protocol ensembles without caching and aggregation capabilities. Due to the low memory, nodes are only able to cache a maximum of 64 foreign chunks, but they mostly evict before they can be utilized by retransmissions, because of the significant chunk flow rate that leads to rapid and frequent cache replacements. The internal chunk buffer is reserved for the respective binary image of the node and is therefore inaccessible by the remaining nodes.

We measure the chunk arrival times for nodes $n_{1...7}$ and demonstrate the update progression in Figure 15. Nodes request 4000 chunks to complete the image delivery, *i.e.,* 28k distinct chunks in total are transmitted on that particular path. The distributions indicate a completion time of ≈30 minutes for the setup with a single device class and a collective binary. On the other hand, the update time considerably decelerates if the NDN multicast features are inactive. Hence, the update process continues for more than two hours when individual binaries are deployed to propagate. The missing hop-wise caching ability means that retransmissions need to traverse the full request path up to the gateway node on each retry, which again promotes higher packet loss probabilities due to the generated side traffic for other, ongoing transmissions. In contrast, in-network caches reduce the number of necessary hops and confine retransmissions in the best case to a single link. The greater slopes towards the end of both *cascading* measurements are an indication that leaf nodes operate quicker with the coordinated retrieval method due to absent nodes in the vicinity that compete for the bandwidth, irrespective of caching abilities.

## 5 CONCLUSIONS AND OUTLOOK

We have studied massive roll-outs of firmware in large-scale constrained multi-hop networks, which is an emerging need but also a major challenge for the IoT edge. We found that information-centric content replication fosters efficient and reliable chunk dissemination, which makes routinely firmware updates feasible even for nodes that are highly constrained in processing power, memory, and radio capacity. Hop-wise forwarding and in-network caching in particular facilitate update campaigns across homogeneous wireless regimes even with intermittent connectivity.

Using the IETF SUIT update model as a blueprint, we further devised and evaluated firmware propagation strategies based on the Named Data Networking (NDN) protocol. We conducted a feasibility analysis using real protocol implementations on a wireless testbed to quantify the effective network performance of retrieving large firmware images in the information-centric Internet of Things. Our findings indicate that (*i*) a simultaneous, uncoordinated distribution of firmwares results in high cross traffic within the broadcast domain that degrades nodal operability, (*ii*) deployments with collective binaries significantly benefit from in-network caching, and (*iii*) a hop-wise, cascading delivery relaxes strain on network resources, allows for continued regular operations during the roll-out process, and preserves limited energy budgets by allowing longer sleep cycles due to prompt firmware deliveries.

This work raises research questions in three directions. First, further insights and optimizations of current design decisions and operational practices are expected to be learned from long-term deployment studies. Second, experiences from massive firmware roll-outs in ICN deployment scenarios may generate valuable feedback for the RESTful, CoAP-centered IoT [15]: Which insights can help to develop the emerging data-centric Web of Things? Third, we propose to explore how content object security [16] can be optimized for the IoT to ease voluminous data transfers without sacrificing integrity, authenticity, and DoS resistance.

## ACKNOWLEDGMENT

We want to thank the anonymous reviewers and our shepherd Alex Afanasyev for constructive feedback and inspiration on how to improve the paper. This work was supported in part by the German Federal Ministry for Education and Research (BMBF) within the projects *RAPstore – RIOT App Store* and the Hamburg *ahoi.digital* initiative with *SANE*.

**A Note on Reproducibility.** We fully support reproducible research [2, 42] and perform all our experiments using open source software and an open access testbed. Code and documentation will be available on Github at https://github.com/inetrg/ACM-ICN-2021-FWUPDATE.

## REFERENCES

[1] Amar Abane, Mehammed Daoui, Samia Bouzefrane, Soumya Banerjee, and Paul Muhlethaler. 2020. A Realistic Deployment of Named Data Networking in the Internet of Things. *Journal of Cyber Security and Mobility* 9, 1 (2020), 1–46.

[2] ACM. Jan., 2017. Result and Artifact Review and Badging. http://acm.org/publications/policies/artifact-review-badging.

[3] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 459–464.

[4] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman. 2012. A Survey of Information-Centric Networking. *IEEE Communications Magazine* 50, 7 (July 2012), 26–36.

[5] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro. 2015. Information Centric Networking in IoT scenarios: The case of a smart home. In *Proc. of IEEE International Conference on Communications (ICC)*. IEEE, Piscataway, NJ, USA, 648–653.

[6] Sobia Arshad, Muhammad Awais Azam, Mubashir Husain Rehmani, and Jonathan Loo. 2019. Recent Advances in Information-Centric Networking-Based Internet of Things (ICN-IoT). *IEEE Internet of Things Journal* 6, 2 (2019), 2128–2158.

[7] Onur Ascigil, Sergi Reñé, George Xylomenos, Ioannis Psaras, and George Pavlou. 2017. A Keyword-based ICN-IoT Platform. In *Proc. of 4th ACM Conference on Information-Centric Networking (ICN)*. ACM, New York, NY, USA, 22–28.

[8] Atmel. 2009. *Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications.* Atmel Corporation. http://www.atmel.com/images/doc8111.pdf

[9] Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (December 2018), 4428–4440. http://dx.doi.org/10.1109/JIOT.2018.2815038

[10] Emmanuel Baccelli, Christian Mehlis, Oliver Hahm, Thomas C. Schmidt, and Matthias Wählisch. 2014. Information Centric Networking in the IoT: Experiments with NDN in the Wild. In *Proc. of 1st ACM Conf. on Information-Centric Networking (ICN-2014)* (Paris). ACM, New York, 77–86. http://dx.doi.org/10.1145/2660129.2660144

[11] C. Bormann, M. Ersue, and A. Keranen. 2014. *Terminology for Constrained-Node Networks.* RFC 7228. IETF.

[12] C. Bormann and Z. Shelby. 2016. *Block-Wise Transfers in the Constrained Application Protocol (CoAP).* RFC 7959. IETF.

[13] Jeff Burke, Paolo Gasti, Naveen Nathan, and Gene Tsudik. 2013. Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control and NDN. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*. IEEE, Piscataway, NJ, USA, 394–398.

[14] Asit Chakraborti, Syed Obaid Amin, Aytac Azgin, Satyajayant Misra, and Ravishankar Ravindran. 2018. Using ICN Slicing Framework to Build an IoT Edge Network. In *Proceedings of the 5th ACM Conference on Information-Centric Networking* (Boston, Massachusetts) *(ICN '18)*. ACM, New York, NY, USA, 214–215.

[15] Cenk Gündogan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. 2020. Toward a RESTful Information-Centric Web of Things: A Deeper Look at Data Orientation in CoAP. In *Proc. of 7th ACM Conference on Information-Centric Networking (ICN)* (Montreal, CA). ACM, New York, 77–88. https://doi.org/10.1145/3405656.3418718

[16] Cenk Gündogan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. 2021. Content Object Security in the Internet of Things: Challenges, Prospects, and Emerging Solutions. *IEEE Transactions on Network and Service Management (TNSM)* (2021). https://doi.org/10.1109/TNSM.2021.3099902

[17] Cenk Gündogan, Peter Kietzmann, Martine Lenders, Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. 2018. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In *Proc. of 5th ACM Conference on Information-Centric Networking (ICN)*. ACM, New York, NY, USA, 159–171. https://doi.org/10.1145/3267955.3267967

[18] Cenk Gündogan, Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. 2020. Designing a LoWPAN convergence layer for the Information Centric Internet of Things. *Computer Communications* 164, 1 (December 2020), 114–123. https://doi.org/10.1016/j.comcom.2020.10.002

[19] Peter Gusev and Jeff Burke. 2015. NDN-RTC: Real-Time Videoconferencing over Named Data Networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking* (San Francisco, California, USA) *(ICN '15)*. ACM, New York, NY, USA, 117–126.

[20] Mahmudul Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. 2013. NLSR: Named-data Link State Routing Protocol. In *3rd ACM SIGCOMM Workshop on Information-centric Networking (ICN '13)*. ACM, New York, NY, USA, 15–20.

[21] J.W. Hui and D. Culler. 2004. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proc. of the 2nd Int. Conf. on Embedded Networked Sensor Systems (SenSys'04)*. ACM, New York, NY, USA, 81–94.

[22] Van Jacobson, Diana K. Smetters, James D. Thornton, and Michael F. Plass. 2009. Networking Named Content. In *5th Int. Conf. on emerging Networking Experiments and Technologies (ACM CoNEXT'09)* (Rome). ACM, New York, NY, USA, 1–12.

[23] S. Josefsson and I. Liusvaara. 2017. *Edwards-Curve Digital Signature Algorithm (EdDSA).* RFC 8032. IETF.

[24] Peter Kietzmann, Lena Boeckmann, Leandro Lanzieri, Thomas C. Schmidt, and Matthias Wählisch. 2021. A Performance Study of Crypto-Hardware in the Low-end IoT. In *International Conference on Embedded Wireless Systems and Networks (EWSN)* (Delft, NL). ACM, New York, USA, 12.

[25] Joel Koshy and Raju Pandey. 2005. Remote incremental linking for energy-efficient reprogramming of sensor networks. In *Proceeedings of the 2nd European Workshop on Wireless Sensor Networks* (Istanbul, Turkey). IEEE Press, Piscataway, NJ, USA, 354–365.

[26] H. Krawczyk, M. Bellare, and R. Canetti. 1997. *HMAC: Keyed-Hashing for Message Authentication.* RFC 2104. IETF.

[27] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. 2002. Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks. *Wireless Networks* 8 (2002), 169–185.

[28] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. 2009. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proc. of the 9th ACM SIGCOMM Conference on Internet Measurement* (Chicago, Illinois, USA) *(IMC '09)*. ACM, New York, NY, USA, 90–102.

[29] Pedro José Marrón, Matthias Gauger, Andreas Lachenmann, Daniel Minder, Olga Saukh, and Kurt Rothermel. 2006. FlexCup: A Flexible and Efficient Code Update Mechanism for Sensor Networks. In *Proceedings of the 3rd European Conference on Wireless Sensor Networks* (Zurich, Switzerland) *(EWSN'06)*. Springer-Verlag, Berlin, Heidelberg, 212–227.

[30] Bertrand Mathieu, Cedric Westphal, and Patrick Truong. 2016. Towards the Usage of CCN for IoT Networks. In *Internet of Things (IoT) in 5G Mobile Technologies*. Springer, Cham, Switzerland, 3–24.

[31] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. 2007. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks.* RFC 4944. IETF.

[32] Brendan Moran, Hannes Tschofenig, and Henk Birkholz. 2020. *An Information Model for Firmware Updates in IoT Devices.* Internet-Draft – work in progress 08. IETF.

[33] Brendan Moran, Hannes Tschofenig, Henk Birkholz, and Koen Zandberg. 2020. *A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest.* Internet-Draft – work in progress 11. IETF.

[34] B. Moran, H. Tschofenig, D. Brown, and M. Meriac. 2021. *A Firmware Update Architecture for Internet of Things.* RFC 9019. IETF.

[35] Marc Mosko. 2016. *CCNx Content Object Chunking.* Internet-Draft – work in progress 02. IETF.

[36] Marc Mosko and Christian Tschudin. 2016. *ICN 'Begin-End' Hop by Hop Fragmentation.* Internet-Draft – work in progress 02. IETF.

[37] Abderrahmen Mtibaa and Spyridon Mastorakis. 2020. NDNTP: A Named Data Networking Time Protocol. *IEEE Network* 34, 6 (September 2020), 235–241.

[38] S. Y. Oh, D. Lau, and M. Gerla. 2010. Content Centric Networking in tactical and emergency MANETs. In *2010 IFIP Wireless Days*. IEEE, Piscataway, NJ, USA, 1–5.

[39] Colin Percival. 2003. *Naive differences of executable code.* Technical Report. daemonology.net.

[40] George C. Polyzos and Nikos Fotiou. 2015. Building a reliable Internet of Things using Information-Centric Networking. *Journal of Reliable Intelligent Environments* 1, 1 (2015), 47–58.

[41] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *IEEE Symposium on Security and Privacy (SP)* (San Jose, CA, USA). IEEE Press, Piscataway, NJ, USA, 195–212.

[42] Quirin Scheitle, Matthias Wählisch, Oliver Gasser, Thomas C. Schmidt, and Georg Carle. 2017. Towards an Ecosystem for Reproducible Research in Computer

Networking. In *Proc. of ACM SIGCOMM Reproducibility Workshop*. ACM, New York, NY, USA, 5–8.

[43] Thomas C. Schmidt, Sebastian Wölke, Nora Berg, and Matthias Wählisch. 2016. Let's Collect Names: How PANINI Limits FIB Tables in Name Based Routing. In *Proc. of 15th IFIP Networking Conference* (Vienna, Austria). IEEE Press, Piscataway, NJ, USA, 458–466.

[44] E. M. Schooler, D. Zage, J. Sedayao, H. Moustafa, A. Brown, and M. Ambrosin. 2017. An Architectural Vision for a Data-Centric IoT: Rethinking Things, Trust and Clouds. In *IEEE 37th Intern. Conference on Distributed Computing Systems (ICDCS)*. IEEE, Piscataway, NJ, USA, 1717–1728.

[45] Wentao Shang, Adeola Bannis, Teng Liang, Zhehao Wang, Yingdi Yu, Alexander Afanasyev, Jeff Thompson, Jeff Burke, Beichuan Zhang, and Lixia Zhang. 2016. Named Data Networking of Things (Invited Paper). In *Proc. of IEEE International Conf. on Internet-of-Things Design and Implementation (IoTDI)*. IEEE Computer Society, Los Alamitos, CA, USA, 117–128.

[46] Z. Shelby, K. Hartke, and C. Bormann. 2014. *The Constrained Application Protocol (CoAP)*. RFC 7252. IETF.

[47] Junxiao Shi and Beichuan Zhang. 2012. *NDNLP: A Link Protocol for NDN*. NDN, Technical Report NDN-0006. NDN Team.

[48] Andy Stanford-Clark and Hong Linh Truong. 2013. *MQTT For Sensor Networks (MQTT-SN) Version 1.2*. Protocol Specification. IBM. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[49] T. Stathopoulos, J. Heidemann, and D. Estrin. 2003. *A remote code update mechanism for wireless sensor networks*. Technical Report 30. Center for Embedded Networked Sensing (CENS), Los Angeles, CA, USA.

[50] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander. 2012. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. IETF.

[51] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. 2014. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys and Tutorials* 16, 2 (2014), 1024–1049.

[52] Yingdi Yu, Alexander Afanasyev, Jan Seedorf, Zhiyi Zhang, and Lixia Zhang. 2017. NDN DeLorean: an authentication system for data archives in named data networking. In *4th ACM Conference on Information-Centric Networking* (Berlin, Germany) *(ACM-ICN '17)*. ACM, New York, NY, USA, 11–21.

[53] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (2014), 66–73.

[54] Minsheng Zhang, Vince Lehman, and Lan Wang. 2016. *PartialSync: Efficient Synchronization of a Partial Namespace in NDN*. Technical Report NDN-0039-1. NDN. https://named-data.net/wp-content/uploads/2016/06/ndn-0039-1-partial-sync.pdf

[55] Zhenkai Zhu and Alexander Afanasyev. 2013. Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking. In *Proc. of the 21st IEEE International Conference on Network Protocols (ICNP 2013)* (Goettingen, Germany). IEEE, Piscataway, NJ, USA, 1–10.

[56] ZigBee Alliance. 2015. *ZigBee Specification*. Specification Document 05-3474-21. ZigBee Alliance. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf