Nebula: A Privacy-First Platform for Data Backhaul

Jean-Luc Watson, Tess Despres, Alvin Tan, Shishir G. Patil, Prabal Dutta, and Raluca Ada Popa
University of California, Berkeley
{jlw, tdespres, alverino, shishirpatil, prabal, raluca.popa}@berkeley.edu

***Abstract*—Imagine being able to deploy a small, battery-powered device nearly anywhere on earth that humans frequent and having it be able to send data to the cloud without needing to provision a network—without buying a physical gateway, setting up WiFi credentials, or acquiring a cellular SIM. Such a capability would address one of the greatest bottlenecks to deploying the long-tail of small, embedded, and power-constrained IoT devices in nearly any setting. Unfortunately, decoupling the device deployment from the network configuration needed to transmit, or *backhaul*, sensor data to the cloud remains a tricky challenge, but the success of Tile and AirTag offers hope. They have shown that mobile phones can crowd-source worldwide local network coverage to find lost items, yet expanding these systems to enable general-purpose backhaul raises privacy concerns for network participants. In this work, we present Nebula, a privacy-focused architecture for global, intermittent, and low-rate data backhaul to enable nearly any thing to eventually connect to the cloud while (i) preserving the privacy of the mobile network participants from the platform provider by decentralizing data flow through the system, (ii) incentivizing participation through micropayments, and (iii) preventing system abuse.**

1. Introduction

Large-scale networks of low-power sensors can enable transformational applications like wildfire monitoring, smart farming [1], search and rescue [2], censorship circumvention [3], [4], and asset tracking [5], [6]. However, *backhauling* (i.e. retrieving) data from widely-dispersed sensors is challenging due to limited options for connectivity.

As a motivating example, consider deploying sensors on urban bike paths to gather usage data. Such data is crucial to inform investments in future infrastructure [7]. Figure 1 shows the parties involved in one approach to data backhaul, inspired by Tile's architecture, to which we refer throughout this paper. The *sensor*, with a low-power BLE radio, gathers usage data. When a *mule* (e.g. a mobile device) is within range, it connects to the sensor and collects a data payload. When the mule returns to network coverage (e.g. cellular or WiFi), the payload is uploaded to an *application server*. Throughout, a *platform provider* manages the backhaul network. While other technologies exist that could transmit data from the sensors deployed in the wide area to the cloud, they suffer from some combination of deployment hassle, high cost, and energy limits.

An ad-hoc data backhaul platform would remove the need for manual data retrieval, application-specific networking infrastructure, or costly and power-hungry wide-area connectivity. Recognizing the benefits of backhaul systems, several bespoke solutions exist, but with limitations. Apple's FindMy network [5] privately reports device location information to device owners, but this system is extremely application-specific. Sidewalk [8] is a static backhaul system operating on Amazon-manufactured hardware. Sidewalk, as a large-scale system, enables new sensor capabilities and extends the range of many existing devices (e.g. Tile is a participant and anyone can add devices [9]). However because it is highly-centralized, Sidewalk collects device identifiers at scale which must be deleted to avoid potentially tracking passers-by [10]. Over time, a centralized provider (in this case Amazon) is able to collect vast amounts of time-location data (tied to device identifier) about the people who participate on the network. Privacy leakage in such a ubiquitous network is a major concern: location data about users has been shown to leak harmful information including medical conditions, religious affiliations, social relationships and location traces [10], [11], [12], [13]. Moreover, none of these systems offer a financial incentive mechanism.

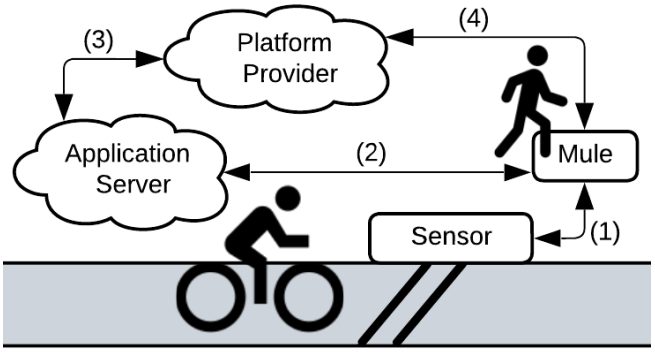

Figure 1: Data backhaul in an urban infrastructure application. Data *mules* (e.g. people with phones) pass *sensors* and collect data (1). When in cellular or Wi-Fi range, they upload the data to an end *application server* (2). A *platform provider* administers the network by charging application servers (3), and paying mules (4).

In this paper, we explore the question: *how do we architect a backhaul system that minimizes the purview of the central platform provider, thereby preserving mule privacy from the provider, while enabling an incentivized, scalable data backhaul network?*

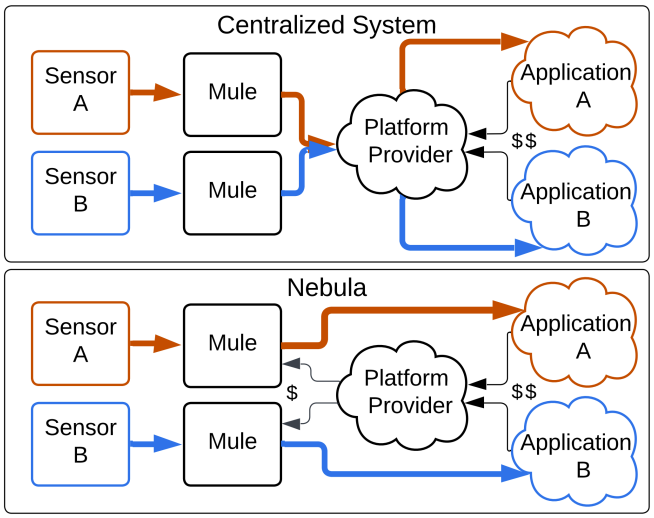

Figure 2: Nebula, unlike centralized designs (e.g. Sidewalk [8]), takes a decentralized approach. Mules route data directly to application servers, improving location privacy.

1.1. Nebula

In response to our design question, we introduce Nebula, a privacy-first platform for general-purpose data backhaul. In contrast to prior backhaul efforts, we prevent mules from revealing their location to the provider, allow the provider to charge applications, compensate mules for system usage, and handle authentication and spam prevention in the absence of end-to-end connectivity. Nebula avoids giving the platform provider wide and deep visibility by employing a decentralized architecture, as shown in Figure 2. At the same time, applications in Nebula can still benefit from the ease of deployment and management that the platform provider offers: the platform provider recruits and manages a network of mules, and handles the associated payments.

Participants in Nebula upload sensor data directly to individual application servers, and interact with the provider entirely *asynchronously* from any data upload, which prevents the provider from observing backhauled data. A decentralized approach allows the platform provider to retain useful properties while eliminating a global view of identity, payment, and location. Removing the platform provider from the data path addresses our primary concern of passive observation of all mule upload behavior, but it also complicates several critical network management functions, which we discuss below. We show that Nebula (a) provides a significantly stronger notion of participant privacy while (b) placing minimal additional burdens on energy- and compute-constrained mobile devices.

Payment. When a provider routes data payloads from mules to application servers, like in Sidewalk [8], billing is simple: charge applications based on forwarded payloads and, if desired, reward mules. The challenge is to maintain this payment functionality while preserving mule privacy. A key insight underlying Nebula is that this process need not require payload-by-payload accounting: applications can pre-purchase tokens and distribute them to mules in exchange for data. Importantly, when redeemed, these tokens should not be linked to the party who purchased them, as this would give the provider information about the applications a particular mule was interacting with. Instead, mules in Nebula redeem these tokens on a fixed schedule, while the tokens themselves are indistinguishable from each other, avoiding additional information leakage.

In this paper, we instantiate our design with PrivacyPass [14], as their token scheme is practically efficient and currently deployed at Internet-scale to automate Internet challenges like CAPTCHAs. PrivacyPass is a building block and by itself not sufficient to provide privacy-preserving backhaul. The original construction assumes that participants will directly exchange tokens with a central server in exchange for resources [14], while Nebula's decentralized setting is significantly different, in that the application servers initially receiving tokens will pass them off out-of-band to various mules who will then attempt to redeem them. Preventing misuse becomes more challenging, requiring a novel protocol that wraps PrivacyPass (Section 5).

Authentication and Spam Prevention. Since mules may backhaul payloads for sensors in areas that lack direct network connectivity, such as in elevators [15], on farmland [1], in parking garages [16], or on hiking trails [17], our system cannot rely on contemporaneous sensor-to-cloud connections for authentication. Instead, we assign persistent identities to sensors, which enable mules to verify sensor validity before accepting data payloads. This conserves mule resources by ignoring unauthenticated or misbehaving sensors. We establish DTLS sessions over BLE connections when mules encounter sensors, and demonstrate low session establishment costs in Section 8.

Handling Misbehavior. In Nebula, mules owned by third-parties directly interact with application servers. This sets up a conflict of interest, in that mules might attempt to abuse the system to gain more compensation without uploading valid payloads, while application servers might try to use mules' uploads without fairly compensating the mules. To address these issues, we present a payload delivery protocol in Section 5.4 that, in the case of incomplete delivery, allows mules to submit anonymous complaints of misbehavior to the platform provider.

Implementation. We implement our protocol design with BLE-enabled sensors (Nordic nRF52840-based) and BLE/WiFi-enabled mules (Espressif ESP32), and evaluate performance in deployment scenarios using real-world BLE data to estimate mule-sensor interactions (Section 8.3). Given the wide range of expected deployment environments, we develop an analytical model of energy and memory consumption Section 6. On average, our results show that sensors are able to upload data at 2.8 kB/s while drawing 40.3 mW (Section 8.1). Based on our measurements, we estimate that a smartphone mule can backhaul 1,000 data payloads every day while only consuming 5% battery each day and 3 MB of storage total (Section 8.3). We deployed application servers and a provider capable of producing and redeeming over 445,000 tokens per second (Section 8.4).

2. Background and Related Work

Low-Power, Wide-Area Networks (LPWANs) are designed to retrieve data for distributed sensor networks, and can be categorized into *licensed spectrum* cellular LPWANs and *unlicensed band* LPWANs. The most well-known and widely-used licensed band standards are NB-IoT and LTE-M. Among LPWANs operating in the unlicensed bands, the most commonly-used protocol is LoRaWAN, an open standard managed by the LoRa Alliance [18] that utilizes the 915 MHz ISM band for communication. LPWANs, while low in power, sacrifice throughput for range. WiFi, on the other hand, has high throughput but is also relatively high power. BLE does not have the range of LPWANs or the throughput of WiFi, but offers the lowest-power and lowest-cost solution [19], [20], [21], [22], [23]. Furthermore, the cost of consumer-focused cellular data plans in the United States has been falling (from $4.64/GB in 2018 to $2.75/GB in 2023) while the cost of cellular IoT remains high [24]. As a result, we argue that the ubiquity of relatively powerful mobile devices (with BLE, cellular and Wi-Fi) have created an environment ripe for low-power, limited-range backhaul from distributed sensors.

2.1. Limited Backhaul Deployments

Helium. Helium is currently the worlds largest LoRaWAN network, with gateways blanketing most European and US urban areas [25]. It crowdsources participants to deploy stationary gateways, mine Helium cryptocurrency (HNT) based on the data they backhaul, and send payloads to Helium routers which forward the data to application servers [26]. Helium uses a Proof-of-Coverage (PoC) algorithm which requires miners to prove that they are providing wireless coverage to a specific region [26]. While Helium lowers the barrier to entry for sensor deployments, it is still limited: many rural areas lack coverage [27], malicious location spoofing is possible [28], and the unlicensed nature of LoRaWAN means that the upload capacity is limited [29]. From a privacy perspective, traffic is logged on a public blockchain where it can be attributed to a particular application [27]. Nebula takes an alternative approach to Helium, performing backhaul opportunistically.

FindMy. In contrast to Helium, Apple's FindMy [5] network represents a vertically-integrated, proprietary backhaul network focused on a single application: location tracking. At frequent intervals, FindMy devices advertize a rotating key to nearby FindMy-enabled Apple devices. The receiving devices use this key to encrypt and upload their own GPS location to a database. The owner of a device can then query for a position report [30] without directly allowing Apple or the other FindMy devices to learn the transmitted position report. Recent work has shown how to build third-party devices that can generate FindMy-compatible keys [31] to piggy-back on the network, and how to transmit a small stream (i.e. tens of bytes per second) of arbitrary information encoded into advertised keys [32]. However, FindMy fundamentally asks sensors to only provide key material, relying on the mules to construct the location report. When transitioning to a general-purpose backhaul network, third-party mules will upload sensor payloads with metadata (i.e. timestamp, destination application) that could leak personal information. In Nebula, a central privacy goal is to limit the information a backhaul platform provider can gain.

Sidewalk. Amazon's Sidewalk network is currently the closest deployed example of a general-purpose backhaul network, enabling Amazon-owned hardware (e.g. Ring doorbells and Alexa smart speakers) to act as Sidewalk Gateways for devices with low-power BLE and LoRA (i.e. 900 MHz) wireless radios [8], [33]. The network has a centralized architecture where endpoint devices connect to a central Sidewalk Network Server (SNS) through Gateways. The SNS then routes data to the appropriate end-destination application server. While this architecture simplifies tasks such as charging for network use and managing access control, it also means that sensors and mules must authenticate directly with the SNS as every connection is made [8]. The routing metadata provided to the SNS includes information on persistent endpoint and gateway identifiers, transmission times, and desired destination application servers. Importantly, this metadata reveals which gateways an endpoint sensor visits, which provides the SNS a centralized view of every device's last-reported location. Nebula, in contrast, ensures routing metadata is not exposed to the central platform provider.

2.2. Related Work

In addition to Helium, Find My and Sidewalk, many smaller-scale wireless networks inform this work. ZebraNet [34], an early wireless network, placed devices on Zebras to track their location. Large scale habitat monitoring offers interesting challenges, such an intermittent connectivity, not seen in end-to-end connected systems [35]. These early attempts, along with others [36], [37], [38], inspired many other deployments [28], [39], [40], [41], [42]. Public Wi-Fi hotspots, including those deployed on trash cans, have been shown to leak information about the people who connect to them [43], [44]. Finally, Google deployed interaction based services with the Physical Web and Eddystone [45], [46], but ran into challenges with spam prevention.

Space or balloon-based sensor networks can also extend connectivity to sensor systems and search and rescue [47], [48], [49], but require expensive infrastructure to scale. Recent public health events have inspired BLE exposure notifications from Apple and Google [50], which maintain participant privacy through the use of exposure keys designed to restrict encounter records to individual user devices. In smart homes, energy data can reveal private information, making data processing-based privacy schemes desirable for smart meters [51]. VPriv [52] and PrivStats [53] provide location privacy when computing functions or statistics on user paths. There are many anonymous messaging protocols (e.g. for whistle blowers or activists) [54], [55], [56], [57], [58], [59], [60], [61], [62], as an alternative to Tor.

2.3. Unlinkable Tokens

Nebula requires an unlinkable token construction that satisfies two security guarantees: *unlinkability* and *one-more-token security*, as defined in Davidson et al. [14]. A scheme's tokens are unlinkable if, when redeemed to a malicious server, the server cannot link the tokens to the client(s) that generated them. Second, a scheme satisfies the one-more-token guarantee if, even with knowledge of many valid tokens, a malicious client cannot forge more valid tokens.

We use PrivacyPass, a protocol originally deployed to privately replace CAPTCHAs in CDNs, in a black-box manner to provide these unlinkable tokens, although Nebula is not tied to this construction. We provide an overview of the protocol below and refer readers to the PrivacyPass papers for a full treatment of the scheme [14], [63].

At its core, the PrivacyPass protocol uses a verifiable oblivious pseudorandom function (VOPRF) [14]. A VOPRF involves a client with some input value x and a server with a PRF secret key k, and calculates the result $t = \text{PRF}_k(x)$ for the client without revealing anything to the server. Later, this result t can be verified as a valid PRF output using a publicly-revealed commitment Y to the secret key k.

A client acquires new unlinkable tokens by picking random inputs and evaluating a VOPRF over them with a central server. Crucially, the server also provides a batched discrete log equivalence proof (DLEQ) to the client, which proves in zero-knowledge that all tokens were signed by the same secret key (the k committed to by Y). For Nebula, this means that we do not even need to trust the provider to generate tokens correctly, as application servers can efficiently check their correctness. Thus, the provider cannot cause privacy leakage by, for example, using many separate PRF keys. Finally, when a mule presents an unblinded token later to the server, the server can efficiently verify the token using the public commitment to the secret key it used during token generation, without being able to identify the mule.

3. System Overview

In this section, we present the Nebula system overview. We begin by highlighting the stakeholders involved, and then discuss the Nebula architecture.

3.1. System Stakeholders

In Nebula, three distinct parties work together to perform sensor data backhaul: the platform provider, application servers along with their deployed sensors, and mules.

The **Platform Provider** is the coordinating entity, hosting cloud infrastructure and managing payment processes at scale. As a system administrator, the provider registers application servers and mule devices and deploys backhaul software to them. As a payment processor, the platform provider charges application providers and compensates mules based on data upload.

Application Servers are entities that wish to deploy sensors and receive data without provisioning their own mule infrastructure. They register with the platform provider and pay based on the volume of data they upload through the mules. Before backhaul begins, they provision their sensors with Nebula-specific credentials (Section 5.1). Any one sensor is managed by only a single application server.

Mules are mobile entities, primarily cell phone users, that have (potentially intermittent) Internet access. They run a Nebula service that detects nearby sensors, collects the sensors' payloads, and delivers them to application servers. They are compensated for their uploads by the platform provider, which incentivizes them to upload data quickly and often. If the compensation is sufficiently large, some mules may choose to act as stationary "routers" around particularly active sensors. In general, smartphones are excellent candidates for mules, as they are mobile, have low-power wireless radios, and frequently connect to the Internet.

3.2. System Architecture

Nebula operates over a series of long-running *epochs* (on the scale of a month). Epochs correspond to points at which the platform provider performs key rotation. The epoch length forms a tradeoff between privacy and quick compensation. To backhaul sensor data to the cloud, Nebula has four main phases shown in Figure 3 and described in Section 5:

1) Application servers *pre-purchase tokens* from the provider, which can be exchanged for sensor payloads.
2) Mules opportunistically encounter deployed sensors, verify their identity, and securely perform *payload pickup* to gather data to backhaul.
3) Mules *deliver payloads* to the desired application server, in exchange for a token purchased earlier.
4) Once an epoch, mules *redeem tokens* with the platform provider and, if they detect misbehavior, can choose to expose it to the provider.

To protect mules' privacy, Nebula's architecture removes the platform provider from the data path, with payloads backhauled by mules directly to the intended application servers without a centralized routing step. Sensor data payloads are end-to-end encrypted between sensors and the application server ensuring that mules and platform providers do not have visibility into data. When picking up packets, mules authenticate sensor certificates by tracing trust up the certificate chain to the root CA (i.e. Nebula Certificate Authority). Similarly, application servers check for packet validity before accepting data and giving tokens to mules, filtering out potential spam. At the end of the epoch, the mules authenticate with the platform provider and trade tokens for payment. This scheme allows us to achieve our privacy goal (Section 4), since the platform provider only sees the number of PrivacyPass tokens it signs for each application server, and the number of valid unlinkable tokens redeemed by each mule, every epoch.

4. Threat Model and Security Guarantees

4.1. System Abuse

Nebula's design includes a large system of third-party devices as backhaul mules, and rewards mule participation through micro-payments. Nebula assumes that mules and

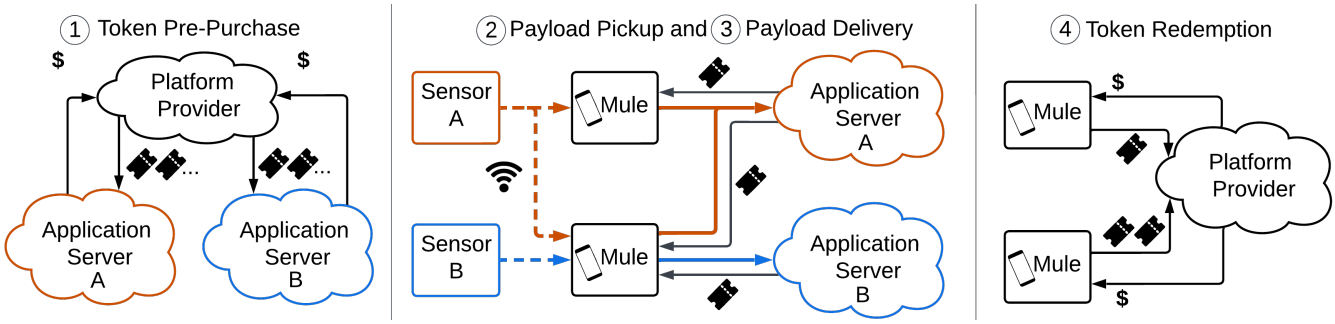

Figure 3: Nebula's privacy-preserving data backhaul architecture. Application servers (1) pre-purchase unlinkable tokens. When mules pass by a sensor, they (2) pick up application payloads and (3) deliver them to the relevant application server in exchange for a token. At fixed intervals (e.g. each month), mules (4) redeem tokens with the platform provider in exchange for micro-payments.

applications servers may behave maliciously for financial gain or to deplete precious network bandwidth, while the platform provider is honest for the purposes of providing soundness: namely, it will not try to render a mule unpaid, or deny service, as it is financially incentivized to continue running Nebula correctly.

Application servers are financially incentivized to minimize the cost of backhaul, so we consider scenarios in which they attempt to cheat mules, either by refusing to exchange tokens or by giving invalid tokens in return for sensor payloads. Similarly, mules might attempt to extract tokens from the application servers without uploading a payload to maximize profit. In Section 5.4 and Section 5.6, we detail Nebula's protocol for payload delivery that allows mules to submit anonymous complaints against misbehavior, but prevents mules from gaining unearned tokens. The mules themselves are third-party devices that might attempt to attack system integrity. We assume that a malicious mule can try to spam other mules with information, or impersonate a sensor for the same purpose. Mules can attempt to collude with each other to replay duplicate payloads or manufacture new ones in the hope of getting paid by the system for participating in an upload. At a high level, Nebula's soundness guarantees are that only valid payloads will result in payment and that payment is provided only once per payload. Through a (rate-limited) complaint process, if the mule correctly followed Nebula's protocol, delivered the payload to an application server, and did not receive payment, the mule will either be able to redeem payment from the platform provider or convince the platform provider that an application server is misbehaving. We state these guarantees along with their proof sketches in Appendix A.

4.2. Privacy

Since backhaul deployments could span millions of personal devices, Nebula's goal is *to preserve mule privacy at the provider* in that it reveals only the following information to the provider each epoch (e.g. each month):

1) How many payloads each mule uploaded system-wide,
2) How many prepaid payload deliveries each application server purchased from the provider, and
3) A set of anonymous complaints against application servers for misbehaving.

At the protocol level, Nebula does not reveal mule identifiers to the provider and application servers during their communications; as for non-Nebula-specific network information that can leak identity (e.g. IP addresses), we rely on complementary mechanisms for anonymizing Internet communication, such as Tor [64], [65] or secure messaging/mixnets [54], [55], [57], [58], [59], [62], [66] that allow clients to anonymize their network metadata when connecting to untrusted servers. In the rest of this paper, we assume that mules can connect to the platform provider and application servers without revealing their identity.

Nebula prevents wide-scale privacy leakage to a service provider by removing the provider's visibility of data payloads containing mule identifiers, destination applications, and encounter timestamps, aggregated over all participants and applications. It is important to note that Nebula's decentralized design does not place additional trust in any application server (AS). Each AS, whether they use a centralized upload service like Sidewalk, or a decentralized service like Nebula, inherently has access to information encoded in their own sensor data payloads, which could include time of upload and sensor location. Thus, ASes may still observe time and location of uploads of their own payloads from the mules as a natural consequence of receiving timely device-specific data from sensors that have been deployed in known locations. In some cases, this can indirectly leak information about mule paths [10] – so in Nebula, the mules are compensated by the application servers for their work as well as for their exposure, and mules choose which application servers they wish to serve.

Collusion. We provide malicious privacy against a platform provider that can collude with other application servers and mules. We consider this strong threat model in particular because nothing stops the platform provider from deploying their own application servers or mules to interact with the remainder of the Nebula deployment.

A platform provider colluding with a subset of application servers will be able to observe all information seen by any of the colluding parties, but not the state of honest application servers or mules. In particular, the malicious

parties will be able to identify the tokens exchanged for any payload (containing time, location, and other application-specific information) uploaded to the colluding application, which they can later link directly to a mule when they redeem the associated token. However, the colluding parties cannot observe actions a mule takes that that do not interact with a corrupted application server, and payloads a mule uploads to an honest server are still unlinkable to the tokens that are eventually redeemed. In the case of a complaint, mules may leak a small additional amount of information as detailed in Section 5.6. This threat model is mirrored in the security definition presented below.

Formalism. To formally define Nebula's privacy guarantee, we use the simulation paradigm of Secure Multi-Party Computation [67]. We provide context for our definition here, and reserve a more detailed treatment, including an explanation of how our simulation-based definition matches the informal guarantee above, for Appendix B.

We refer to an experiment called the *real world*, which contains parties running the actual Nebula protocol, and an *ideal world*, which encodes what the adversary $\mathcal{A}$ learns in a privacy-preserving backhaul system. In both worlds, we consider what information leaks when executing all possible sequences X of Nebula operations (i.e. token purchase, payload delivery, token redemption, complaints, and epoch changes). Below, we define what it means for Nebula to be privacy-preserving, where λ is the security parameter.

Definition 1 (Privacy-Preserving Backhaul System)**.** *Let π be the protocol for a backhaul system, providing parties with the API defined in Section 5.*

We say that π is privacy-preserving if there exists a non-uniform probabilistic polynomial-time machine $\mathcal{S}$ such that, for every non-uniform probabilistic polynomial-time machine $\mathcal{A}$, and for every valid sequence X:

$$\{\text{REAL}_{\pi,\mathcal{A}(z),X}(1^\lambda)\}_\lambda \stackrel{c}{\equiv} \{\text{IDEAL}_{\mathcal{S},\mathcal{A}(z),X}(1^\lambda)\}_\lambda$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability.

Theorem 1 (Privacy in Nebula)**.** *Assume a semantically secure encryption scheme, a existentially unforgeable signature scheme, a collision-resistant hash function, and a simulator for Unlinkable Tokens $\mathcal{S}_{UT}$ in Definition 2. Then π_{Nebula} is privacy-preserving as defined in Definition 1.*

Due to space constraints, we relegate the remaining formalism and proof sketches to Appendix B.

4.3. Limitations

Nebula does not consider the existence of a number of orthogonal sources of leakage and misbehavior that can be mitigated through existing means. For example, the deployed application sensors may attempt to glean identifying information from any data a mule wirelessly broadcasts (e.g. physical chipset irregularities [68] or unique advertised information [69]). These problems are not unique to Nebula – local wireless device tracking and identification has seen significant recent interest [70], [71], [72], and progress in that area can complement a Nebula deployment.

Given that they are mostly deployed in uncontrolled areas, sensors are likely to face physical compromise. Malicious sensors might attempt to impersonate other sensors or mules, tying up computational and communication resources by constantly advertising new or malformed payloads. Misbehaving sensors can be blocked by their current MAC address, and in extreme cases, a mule can simply turn off their radio until they move to a different physical location. However, since backhaul systems are by nature best-effort, this work does not consider denial of service attacks with limited local impact. Similarly, just like any other deployed internet-connected service, Nebula cloud parties (provider and application servers) could experience DDoS attacks, which can be remedied with complimentary solutions proposed in [73], [74], [75]. In line with *best-effort* backhaul, if mules choose to drop data or never connect to Wi-Fi, application servers could lose some data, but in this case mules will not be paid. To add more reliability, application servers can choose to pay for duplicate data packets.

5. Protocol

In this section, we outline the Nebula protocol. This includes device provisioning and deployment, token pre-purchase, payload pickup and payload delivery along with epoch updates and complaint management.

5.1. Provisioning and Deployment

Application servers are provisioned with a public-private keypair pk_{as}, sk_{as}, a matching certificate for pk_{as}, $c_{pk_{as}}$, signed by the platform provider, and an AES symmetric key k_{comm} preshared with the platform provider. The asymmetric keypair is used to attribute messages to the application server when delivering a payload, and k_{comm} protects the confidentiality and integrity of token commitments made by the application server for the platform provider. The application servers check $c_{pk_{as}}$, and additionally use Certificate Transparency [76] to prevent impersonation. Mules also check these certificates against Certificate Transparency.

Each application server provisions and deploys each of their sensors s with a unique sensor ID id_s, an AES symmetric key k_s, a public-private keypair pk_s, sk_s, and a matching certificate for pk_s, c_{pk_s}, that is signed by the application server.

The sensor ID id_s is used by the application server to identify the source of incoming payloads, and k_s secures end-to-end payload confidentiality and integrity between the sensor and the application server using AES-based authenticated encryption with additional data (AEAD). k_s can be securely derived from id_s and a secret key known only to the application server, such that the application can easily and quickly derive k_s from the sensor ID. In the event that a sensor needs a new key, the application server can assign a new id'_s to the sensor, but must physically redeploy it with the new secret key. pk_s and c_{pk_s} are used by nearby mules

Algorithm 1 Token Purchase

Input: Vector of n_t randomly sampled blinded tokens
Output: Vector of signed tokens T s.t. $|T| = n_t$ or $\perp$
Participant(s): Between App Server (AS) and Provider (P)

1: AS pays P to purchase n_t tokens and P updates its count of tokens purchased by AS in the current epoch.
2: AS randomly samples a vector of n_t values T' and performs $t = \mathsf{PrivacyPass.Sign}(t')$ with P using the delivery keypair on each value t' in T'.
3: If the signing protocol succeeds, AS obtains a vector of signed tokens T s.t. $|T| = n_t$, else $\perp$.

to locally authenticate the sensor, establish a secure wireless connection, and verify membership in a specific deployment.

Finally, each epoch, mules are granted a small, fixed number of t_c *complaint tokens*, which can be verified by the platform provider using a different PrivacyPass keypair than normal data upload tokens (they are not interchangeable). These tokens limit the number of complaints an individual mule can lodge against application servers each epoch; above this limit, mules should stop interacting with particularly malicious application servers.

5.2. Token Pre-Purchase

Before the mule can upload the sensor data to the application server, the application server must pre-purchase tokens that can be exchanged with mules for delivering sensor payloads as in Algorithm 1. To prevent the platform provider from identifying which application servers each mule sent data to (which may be associated with mule activity or location information), the application server generates a large set of signed tokens by executing the PrivacyPass signing protocol (PrivacyPass.Sign, Section 2.3) with the platform provider, for some monetary cost over an encrypted TLS sessions.

5.3. Payload Pickup

When a sensor s desires to upload sensor data, it broadcasts wireless advertisements with a Nebula-specific identifier to nearby devices. Any mule that passes by can then identify and choose to connect to the advertising sensor. To confirm that the sensor is authorized to send data using Nebula, the sensor authenticates to the mule while establishing a TLS session, from which following communication over the wireless connection derives confidentiality and integrity. For example, malicious nodes in the same location cannot inject traffic into the wireless link. Most importantly, the sensor identifies itself to the mule using its certificate c_{pk_s}, which the mule can use to confirm that the sensor belongs to an application server for which it has chosen to backhaul data. The mule does not mutually authenticate with the sensor to prevent directly leaking its identity to the application. If session setup fails, the mule ignores the sensor and closes the wireless connection.

If authentication is successful, the sensor generates data payload and end-to-end encrypts it using authenticated symmetric encryption with a fresh nonce under key k_s to create

Algorithm 2 Payload Delivery

Input: Application identifier (id_{AS}), data (d), payload hash (P_{hash}), and signature (σ_{hash})
Output: Tokens and complaint record (t, c_t) or $\perp$
Participant(s): Between Mule (M) and App Server (AS)

1: M creates an anonymous encrypted channel with AS identified by id_{AS}.
2: M sends $P_{hash} = (H(d), \text{id}_s)$ and $\sigma_{hash} = \mathsf{Sign}(P_{hash}, sk_s)$ to AS.
3: AS checks the sensor's id and $\mathsf{Verify}(P_{hash}, \sigma_{hash}, pk_s)$, as well as that there are no duplicate payloads matching $H(d)$, aborting if both checks do not succeed.
4: AS samples a random nonce r and unused token t.
5: AS encrypts the token $\hat{t} = \mathsf{Enc}(t, sk_{comm})$ using a token commit secret key sk_{comm} shared by AS and P, then sends pre-delivery payload $P_{pre} = (r, \hat{t}, H(d))$ and signature $\sigma_{pre} = \mathsf{Sign}(P_{pre}, sk_{as})$ to M.
6: M checks $\mathsf{Verify}(P_{pre}, \sigma_{pre}, pk_{as})$ succeeds, or aborts.
7: M sends d to AS.
8: AS verifies $H(d)$ matches the payload hash in P_{hash}, or aborts. AS sends the unencrypted token in $P_{token} = (r, t, H(d))$ and $\sigma_{token} = Sign(P_{token}, sk_{as})$ to M.
9: M checks $\mathsf{Verify}(P_{token}, \sigma_{token}, pk_{as})$ succeeds, or aborts.
10: On success, M retains token t and a complaint record $c_t = (P_{pre}, \sigma_{pre}, P_{token}, \sigma_{token})$. If the protocol did not complete, M obtains an empty token and complaint record $c_t = (P_{pre}, \sigma_{pre}, d)$, and adds c_t to its complaint list C. If the protocol aborts, the M obtains $\perp$.

d. This encryption ensures that only the correct application server can verify and decrypt the sensor data payload, and provides confidentiality and integrity against tampering by other parties that obtain the payload. The sensor also generates a header payload P_{hash}, containing id_s and a hash of the payload H(d), and signs the header with its secret key, yielding signature σ_{hash}. This hash can allow the application server to identify when duplicate payloads are being submitted (Section 5.5). The sensor then sends the mule P_{hash}, σ_{hash}, and d. Once a mule has received a complete backhaul payload – P_{hash}, σ_{hash}, and d – it is ready to deliver the payload to the application server.

5.4. Payload Delivery

To deliver a payload, the mule forms a TLS connection through an anonymous communication service with the destination application server. Before accepting a payload, the application server verifies the hash of both the payload P_{hash} and signature σ_{hash}. If the payload is valid, the application server picks a token t it will exchange for a valid data d upload. The payload delivery scheme is shown in detail in Figure 4 and Algorithm 2.

During the exchange, payloads may be malformed, so the mule and app servers must carefully verify every payload and signature. We also identify three non-trivial cases where misbehavior during the delivery phase could result in an

Algorithm 3 Token Redemption

Input: Tokens accumulated T
Output: $T_{invalid}$ or $\perp$
Participant(s): Between Mule (M) and Provider (P)

1: M authenticates with P and sends token list T
2: For each t in T, P checks that (1) PrivacyPass.Verify(t) succeeds and (2) t is not yet in P's redeemed token list. On success, P adds t to its list of M's redeemed tokens. On failure, P adds t to invalid token list $T_{invalid}$. If a duplicate, P adds t to the original redeemer's T_{dup} list.
3: P sends $T_{invalid}$ to M, or $\perp$ if the verification aborts.
4: For each invalid token, M adds the corresponding complaint record retained from payload delivery (Algorithm 2) to its complaint list C.

application server not receiving a payload or a mule not receiving a valid token in exchange (Figure 4), which require an out-of-band *complaint* process. In particular, (1) a mule may simply decide not to deliver payload d after receiving P_{pre}, (2) an application server might not respond with any token t, or (3) it might respond with an invalid token. We show in Section 5.6 that a separate complaint process can force a mule to upload missing payloads in the case of (1) and confirm application server misbehavior to the platform provider in the case of (2) or (3).

5.5. Token Redemption

Once per epoch, each mule forms a secure TLS session with the platform provider, authenticates itself, and redeems its accumulated tokens. The platform provider verifies each token has been correctly signed using PrivacyPass.Verify and indicates any invalid tokens to the mule. The server must then verify that the valid tokens are not duplicates of redeemed tokens, in order to prevent replaying tokens for economic gain. As immediate remuneration is not critical, Nebula allows the platform provider to check for replays in the background, and notifies mules at the end of the epoch if any of their tokens have seen duplicate submissions. Finally, the provider compensates mules out-of-band for the number of valid, unredeemed tokens they submit.

In order to prevent the database of already-submitted tokens from getting too large, we flush old token reuse databases from previous epochs when the platform provider rotates keys each epoch. The Nebula platform provider retains the token database and key material from the most recent previous epoch so that old tokens can still be redeemed by mules across one epoch boundary, but rejects any older tokens. Application servers are responsible for only distributing tokens corresponding to the current epoch, otherwise, mules may submit complaints to the platform provider after having receiving invalid tokens.

5.6. Complaining About Misbehavior

We address the significant latitude for issues during payload delivery by allowing mules to register complaints with the

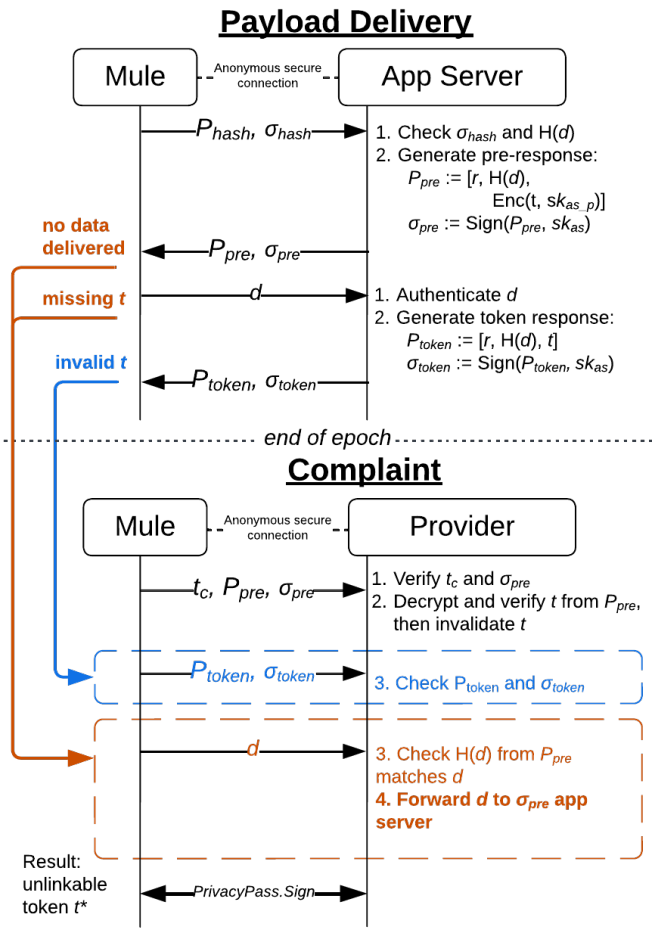

Figure 4: The Nebula delivery (top) and complaint (bottom) protocols. Using a complaint token t_c, a mule can submit a complaint to the platform provider alleging misbehavior if the exchange is not completed (orange), or if the token is invalid (blue). If valid, the platform provider grants the mule a new token and forwards the missing payload to the application. The mule reveals nothing in the complaint other than that it interacted with the application in that epoch.

platform provider, once per epoch, with the hope of recovering a token lost to misbehavior. Note that to do so, mules must leak a small amount of information – they interacted with the application server they are accusing at some point in this epoch and the size of the payload – privacy-conscious mules retain the ability to never avail themselves of the complaint process. For the platform provider, complaints against specific applications can be fed into a rating system that can identify misbehavior over time and allow the provider to take action. To complain, the mule uploads one of its limited complaint tokens and information about the delivery interaction for the platform provider to evaluate at the start of an epoch as described in Algorithm 5. Tokens provided by honest ASes will not collide, but dishonest ASes that collude to reuse tokens will be identified as malicious during complaint process. The bottom section of Figure 4 and Algorithm 4 detail the complaint process.

Algorithm 4 Complaint

Input: A single complaint token (t_c) and record (c)
Output: A valid token for redemption t^* or $\perp$
Participant(s): Mules execute on the start of a new epoch

1: M creates an anonymous encrypted channel with P and sends complaint token t_c and complaint record c_t generated by payload delivery (Algorithm 2).
2: P checks that $\mathsf{PrivacyPass.Verify}(t_c)$ succeeds with the current epoch complaint keypair, t_c is not yet in P's used complaint token list, $\mathsf{Verify}(P_{pre}, \sigma_{pre}, pk_{as})$ succeeds using the σ_{pre} in c, and $\mathsf{PrivacyPass.Verify}(t)$ succeeds with the current epoch delivery keypair, for the decrypted token $t = \mathsf{Dec}(\hat{t}, sk_{comm})$
3: On failure, P sends $\perp$ to M. On success, P adds t_c to its used complaint token list and adds t to its redeemed token list to prevent duplicate redemptions.
4: If c contains a P_{token} payload, M sends P_{token} and σ_{token} to P, who checks that $\mathsf{Verify}(P_{token}, \sigma_{token}, pk_{as})$ succeeds and the token in P_{token} is actually invalid or a duplicate, and aborts if not. Sends $\perp$ if a mule has already complained about this token as a duplicate.
5: Otherwise, M sends d to P, who verifies that $H(d)$ matches the hash in P_{pre} and forwards d to the AS that signed σ_{pre} if successful and aborts if not.
6: M samples a random value $t^{*'}$ and performs $t^* = \mathsf{PrivacyPass.Sign}(t^{*'})$ with P using the next epoch delivery keypair.
7: If the signing protocol succeeds, M obtains a token t^* redeemable in the new epoch, else $\perp$.

Algorithm 5 New Epoch

Input: Complaint count n_c, prior epoch T_{dup} token list
Output: Complaint tokens and valid tokens (T_c, T^*) or $\perp$
Participant(s): Mules execute on the start of a new epoch

1: Each M marks the next epoch active, randomly samples a vector of n_c values T'_c and performs $t_c = \mathsf{PrivacyPass.Sign}(t'_c)$ with P using the next epoch complaint keypair on each value t'_c in T'_c.
2: If the signing protocol succeeds, each M obtains a complaint token list T_c s.t. $|T_c| = n_c$, else $\perp$.
3: M adds complaints matching tokens in T_{dup} to C.
4: Each M anonymously runs the complaint protocol (Algorithm 4) with each of the complaint payloads in its C, obtaining a set of new tokens T^* valid in the new epoch, else $\perp$.

6. Analytical Model

By nature of intermittent interactions between sensors and mules, the memory and energy consumption of sensors and mules in such a system may vary dramatically across applications and locations. In an effort to characterize the main factors that affect the performance of this system and to make informed sensor design choices, we develop two numerical models – one for a deployed sensor and one for a mule – and explore how memory and energy consumption may vary depending on the deployment. In Section 8.3, we use these models to estimate the lifetime of a sensor and consider how many payloads a smartphone mule can upload given energy and memory constraints.

6.1. Sensor Model

Low-power sensors provisioned with BLE are often designed to run for years while periodically advertising [77]. In this section, we explore sensor performance in Nebula.

Sensor Configuration. We assume that the sensor runs off of a battery of size $size_{battery}$ and employs some sensing workload $wkld$ (i.e. periodic sensing, event-driven sensing, or long-running sensing). The sensor accumulates data and desires to upload the data at a frequency of f_{upload} (e.g. once a day). When the sensor desires to upload data, it starts broadcasting BLE advertisements at a rate of f_{adv} until it successfully forms a connection with a nearby mule. Once the connection is made, the sensor stops broadcasting BLE advertisements.

External Conditions. The largest source of uncertainty for the sensor is in how long it must broadcast BLE advertisements before forming a connection. The duration of this time can be parameterized by mule arrival frequency $f_{arrival}$, how long each mule stays in the vicinity of the sensor τ_{mule}, how frequently each mule listens for advertisements f_{listen}, and the probability that any given attempt at connecting with a mule transfers all the data $p_{success}$. These values vary greatly depending on the physical location, time of day, or time of year. While we explore some of the range that these values can take on in Figure 7 and Section 8.3, for the sake of this model, we make the simplifying assumption that these values are constant and the interactions are uniformly distributed.

Sensor Energy Consumption and Lifetime. The average power consumption of the sensor can be broken into two parts: workload power from sensing, and network power from BLE advertisements and the wireless communication Nebula requires. We denote the workload power P_{wkld} and calculate the network power. At any moment in time, we assume there are $f_{arrival} * \tau_{mule}$ number of mules near the sensor. Since each mule listens at a frequency of f_{listen}, the period of time between is $1/(f_{arrival} * \tau_{mule} * f_{listen})$. If it only takes one listening event to successfully connect to an advertising sensor, a sensor has to advertise for $1/(2{*}f_{arrival}{*}\tau_{mule}{*}f_{listen})$ amount of time on expectation before connecting to a mule. A sensor must connect to an average of $1/p_{success}$ mules before successfully handing off a data payload. Thus, the average networking power is $P_{ntwk} = \left(\frac{f_{adv}*E_{adv}}{2*f_{arrival}*\tau_{mule}*f_{listen}} + E_{conn}\right) * \frac{f_{upload}}{p_{success}}$ where E_{adv} is the energy consumed per BLE advertisement and E_{conn} is the energy consumed while uploading data to the mule. This gives us a sensor lifetime of $T_{sensor} = \frac{size_{battery}}{P_{wkld}+P_{ntwk}}$.

6.2. Mule Model

We estimate the energy and memory consumption on a mule for participating in Nebula.

Mule Configuration. We assume that each mule listens for BLE advertisements at a rate of f_{listen} (e.g. 0.1 Hz), uploads the collected data payloads at a rate of f_{upload}, and redeems the tokens at a rate of f_{redeem}. We assume that the mule does this perpetually and uniformly across time.

External Conditions. The largest uncertainty for the mule is how many sensors it will encounter and how much data each sensor will try to backhaul. We parameterize these by expected sensor interaction frequency $f_{interact}$ and expected sensor payload size $size_{payload}$, and assume that these values are constant and the interactions are uniformly distributed. In reality, interactions with sensors may be bursty and payload size variable, however assuming constant values allows us to compute a reasonable estimate.

Mule Energy and Memory Consumption over Time. A mule is expected to periodically do three distinct tasks: collect data from sensors, upload data to servers, and redeem tokens with the platform provider. We consider the *energy consumption* of each of these tasks separately.

$P_{collect} = f_{listen} * E_{listen} + f_{interact} * E_{interact}$, where E_{listen} is the energy consumed each time the mule listens for BLE advertisements, and $E_{interact}$ is the energy consumed while receiving data from a sensor. This happens perpetually throughout the day, and so consumes battery at a rate of $\frac{P_{collect}}{size_{battery}}$.

$P_{upload} = f_{interact} * E_{upload}$, where E_{upload} is the energy consumed each time the mule uploads a single payload. If data is continually uploaded throughout the day, this would consume battery at a rate of $\frac{P_{upload}}{size_{battery}}$. If the upload is batched instead, each time the mule uploads consumes $E^{batched}_{upload} = \frac{P_{upload}}{f_{upload}} = \frac{f_{interact}}{f_{upload}} * E_{upload}$, consuming $\frac{size_{battery}}{E^{batched}_{upload}}$ of the battery. As the upload schedule is flexible, the uploads could be delayed and batched to correspond with mule recharging patterns.

$P_{redeem} = f_{interact} * E_{redeem}$, where E_{redeem} is the energy consumed each time the mule redeems a single token with the platform provider. If tokens are redeemed perpetually throughout the day, this would consume battery at a rate of $\frac{P_{redeem}}{size_{battery}}$. If the redemption is batched instead, each time the mule redeems consumes $E^{batched}_{redeem} = \frac{P_{redeem}}{f_{redeem}} = \frac{f_{interact}}{f_{redeem}} * E_{redeem}$ amount of energy, amounting to $\frac{size_{battery}}{E^{batched}_{redeem}}$ amount of the battery. Since the redemption frequency should be very low in order to obfuscate mule timing information from the platform provider, this would preferably be delayed and also scheduled to correspond to mule recharging patterns.

Mule *memory consumption* consists of data payloads picked up from sensors that haven't yet been uploaded and tokens received from the servers that haven't yet been redeemed. We assume that the mule deletes sensor payloads after uploading them to the servers, and tokens after redeeming them with the platform provider. Suppose that a mule starts with zero memory consumption at time $t = 0$. The expected memory consumption of a mule over time is $M_{mule}(t) = f_{interact} * \left[size_{payload} * \left(t \mod \frac{1}{f_{upload}} \right) + size_{token} * \left(t \mod \frac{1}{f_{redeem}} \right) \right]$ where $size_{token}$ is the size of each token. Then the maximum memory consumption is $f_{interact} * \left(\frac{size_{payload}}{f_{upload}} + \frac{size_{token}}{f_{redeem}} \right)$.

7. Implementation

We describe our Nebula prototype that we implemented to test the overall performance of the backhaul system.

Hardware and Setup. We implement each sensor on an nRF52840 development board, which is provisioned with a 256-bit AES key shared with the sensor's application server, public-private `secp256r1` key pair, and a matching ECDSA certificate signed by the application server. Our prototype mule is a ESP32-WROOM development board. Both the platform provider and the application server are implemented as GCP instances.

Token Pre-Purchase. In order to allow application servers and the platform provider to generate signed unlinkable tokens, we wrap a Rust-based implementation of the PrivacyPass protocol [14], [78] (Section 2.3) with Python bindings and instruct application servers to pre-purchase tokens in 100-token chunks when they have exhausted all of their previously-purchased tokens.

Payload Pickup. Our prototype sensors upload data packets over a BLE connection to mules they encounter over a DTLS secure session. We implemented DTLS sessions over BLE because session establishment requires the mule to verify the correctness of the sensor's certificate without internet connectivity, and results in a secure channel. In addition DTLS sessions can be established without requiring the user to input a code or push a button, as is the case in BLE Secure Connections [79].

The sensor advertises that it has data for pickup while the mule board scans for sensors advertising a Nebula BLE service. Once connected, the Nordic-based sensor acts as a BLE *peripheral* and the ESP32 mule acts as a BLE *central*. We structure the Nebula BLE service with two characteristics (e.g. writable and readable attributes). We take advantage of the two characteristics to create read and write "sockets" for `MbedTLS`. As BLE uses notifications to indicate characteristic changes, we carefully manage state indicating whether each party is listening, receiving, or writing in order to synchronize the parties. During the DTLS handshake, the mule verifies that the sensor owns a certificate that has been signed by the Nebula Certificate Authority (CA) hierarchy. Specifically, a successful DTLS session setup ensures that the sensor certificate is signed by its application server, acting as an intermediary CA for the platform provider, who is the root CA. Once verified, the sensor is able to send its payload to the mule. The payload itself is end-to-end encrypted with AES-GCM, using 12-byte nonces and 16-byte authentication tags.

Payload Delivery. Sensor payloads are stored in the ESP32-based mule's memory until the mule comes within WiFi range of a known network. At that point, a TLS session is initialized with the correct application server and the received payloads are uploaded. The application server attempts to perform AES-GCM decryption on the

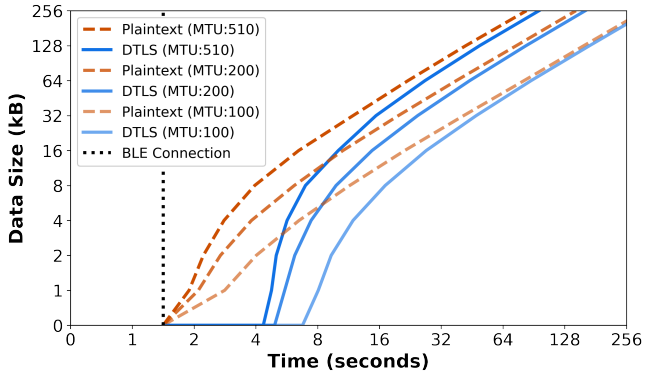

Figure 5: The amount of data that can be transferred based on how long a mule is in connection range with different BLE MTU sizes. Handshake time is amortized as mules spend longer in proximity to sensors.

payload using the key derived from the payload's sensor ID and its own secret key (Section 3.2) and returns a signed PrivacyPass token if successful.

Token Redemption. Our platform provider is conveniently packaged as a container, making deployment easy. We run the provider on a 128-core, general-purpose GCP virtual machine (`n2-standard-128`). The system utilizes two persistent databases: `token_db`, which checks for duplicate entries in the current epoch's submitted tokens, and `mule_payment_db`, which records mule upload totals. Our front end HTTPS server is hosted using *uvicorn*, a popular open-source solution, ensuring connections for mules and application servers. We configure mules to upload in batches of 700 tokens per request. Upon mule token redemption, we first verify the signature and submit the tokens to `token_db`, an in-memory hash-table. To optimize performance, we divide `token_db` into 16 shards. Backend workers then update `mule_db`, a SQLite database that tracks each mule's owed amount for out-of-band payments. To keep costs low, in our prototype, we hosted `token_db` and `mule_payment_db` on the same node as our HTTPS provider, but can be easily replaced with a hosted in-memory and on-disk databases, to facilitate horizontal scaling.

8. Evaluation

We built and evaluated Nebula to demonstrate the feasibility of developing a privacy-preserving, large-scale data backhaul system and answer four key questions:

1) What are the energy costs between a mule and a sensor? What is the overhead of implementing security?
2) What energy costs does a mule incur in delivering sensor data and redeeming tokens?
3) What is the expected energy and memory consumption of running the Nebula service?
4) How well can Nebula's cloud-based provider perform?

8.1. Payload Pickup

Sensor to mule data uploads are by nature transient. Therefore, we measured the amount of data that can be transferred based on how long a mule is near (within BLE range) of a sensor Figure 5. Since transfer rate is highly dependant on BLE implementations, we picked three different BLE maximum transmit units (MTUs) to measure with. BLE has to connect the devices before data can be transferred, this overhead is shown in the black dotted line in Figure 5. We conservatively measured the BLE connection time by including the task start and subscription to BLE characteristics. Encrypted DTLS has both the connection startup time (approximately 1.5 seconds) and TLS handshake time (approximately 2.5 to 5 seconds depending on MTU size) as overhead. However, as more data is transferred the handshake overhead of a few seconds is amortized. We expect that the BLE link could be further optimized, which would improve both the handshake time and data upload time.

Additionally, energy usage on the sensor device is important for prolonged battery use. We measured current draw on the nRF52840 while advertising to be 5.7 mA and current while transmitting to be 12.2 mA. The board supply power was 3.3 V, therefore the advertising and transmitting power is 18.81 mW and 40.26 mW, respectively. Figure 6 shows the breakdown of energy spent in different phases (connection, handshake, and transmit). All current measurements were taken with a Keithley SMU 2401 source meter.

8.2. Payload Delivery

In our architecture, the mule has a larger battery capacity compared with the sensor. However, the overall cost to upload data is still important. We measured the time taken for our ESP32 mule to set up an HTTPS connection with and send HTTPS requests to our application server and our platform provider. We found that the transmission time is largely dominated by the time it takes to set up a connection and the round-trip time of an HTTPS request, more so than the size of each request. In particular, the handshake time is around 2.4 ± 0.3 seconds and each subsequent round trip request took 0.8 ± 0.1 seconds. However, once the size of a request exceeded around 10 kB, we started seeing latency increasing with packet size, which we believe is due to queuing delays from filling up the ESP32's WiFi transmission buffer. We also measure the current draw on the ESP32. We found on the ESP32 development board, as expected, that the WiFi radio draws more power (454 mW) on average compared with the BLE radio (225 mW). Thus, uploading a single packet requires 3.2 seconds and 1.45 J.

Smartphone energy usage. In our evaluation, we implement the mule on a development board in order to isolate the execution of Nebula's protocol from other confounding factors in phone operating systems. In particular, the difficulty of evaluating long-running behavior when backgrounded and issues with OS Bluetooth networking stacks, without OS-level support, require significant engineering "hacks" as described in [80] that are beyond the scope of this work. However, the WiFi and BLE chipsets in modern phones are significantly more power-optimized than that on the ESP32 that we evaluate. To give a comparison point between our implementation and expected energy usage in a smartphone

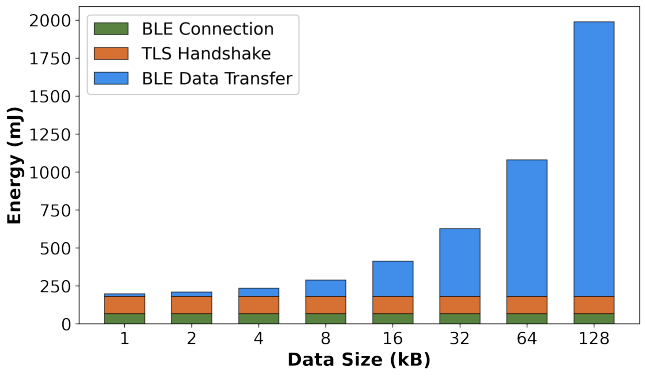

Figure 6: The energy used by the nRF52840 for different payload sizes. There is a set amount of energy spent on setup, so the larger the payload the more the energy is amortized.

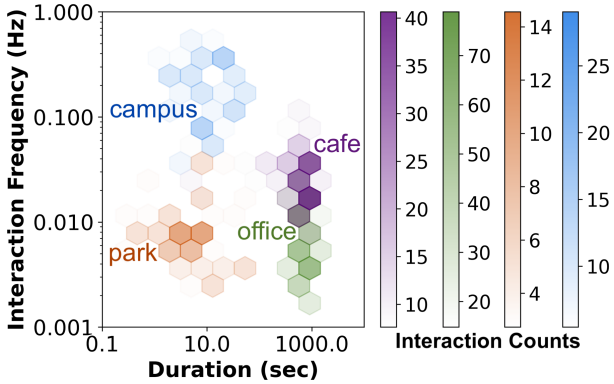

Figure 7: Interaction frequency and interaction duration varies depending on the location. We collect BLE advertisements in four representative locations and construct interactions from repeated MAC addresses. For this data collection, we anonymized all MACs with an irreversible hash and received an IRB exemption from our institution review board.

deployment, we profiled WiFi/BLE power usage on a Pixel 7 Pro running Android 13 using the integrated On Device Power Monitor [81]. While uploading data over WiFi to our application server, the smartphone draws 116 mW on average, 25% of the ESP draw. Sensor data transfer over BLE required 50 mW on average, 22% of the ESP required power. As a result, the energy usage analysis in our evaluation represents a conservative overestimate of what would likely occur in a widespread smartphone deployment.

8.3. Energy & Memory Usage Estimates

We now apply our energy consumption data from Section 8.1 and Section 8.2 to an estimate of sensor and mule behavior, using our analytical model from Section 6.

Sensors. We consider sensors deployed for the bike counting example presented in Section 1. Suppose each sensor counts the number of bicyclists that pass by it every hour and draws around $P_{wkld} = 50$ μW doing so [77]. Each sensor desires to upload its data once a week, so $f_{upload} = \frac{1}{604800}$ Hz. Each payload consists of $24 * 7 = 168$ samples, each containing a timestamp and a bicyclist count, yielding payloads on the order of 1 kB.

Next we check if, given the deployed location, nearby mules are likely to stay in the sensor's vicinity long enough to pick up the data packets. We reference Figure 5, which shows that it would take about 5 seconds to connect to a mule and transfer 1 kB of data. Figure 7 suggests that areas with foot traffic (e.g. on campus and in a park) will contain many mules that are in BLE range of a sensor for at least 5 seconds, so we should be able to successfully upload data to a mule. However, to provide a conservative estimate and account for interactions which are shorter than 5 seconds let's suppose that $p_{success} = 0.5$. From these clusters of mule interactions, the expected duration is around $\tau_{mule} = 10$ seconds with a mule arrival rate of $f_{arrival} = 0.01$ Hz for sparser areas, again estimated from Figure 7.

Using our BLE advertising power $f_{adv} * E_{adv} = 18.81$ mW and supposing mules listen at a rate of $f_{listen} = 0.1$ Hz, we get $P_{ntwk} = \left(\frac{18.81 \times 10^{-3}}{2 * 0.01 * 10 * 0.1} + 0.0169\right) *$ $\frac{1/604800}{0.5} = 3.2$ μW, which is an order of magnitude lower than the workload power. Given a coin cell battery of $size_{battery} = 2200$ J [82], the sensor would last $T_{sensor} = \frac{2200}{53.2 \times 10^{-6}}$ seconds ≈ 1.3 years. Given two alkaline AA batteries with a total capacity of $size_{battery} = 27000$ J [83], the sensor would last $T_{sensor} = \frac{27000}{53.2 \times 10^{-6}}$ seconds ≈ 16 years.

This is only one simplified example, and meant to demonstrate that there's nothing fundamentally infeasible about deploying an application on Nebula. Different applications would have their own challenges and opportunities. For instance, some sensors may want to advertise perpetually (e.g. for global asset tracking), while others would be able to coordinate backhaul times with collaborating mules (e.g. in smart farming). This section is only a starting point from which a developer can consider how their own sensors would interact with the Nebula system.

Mules. Consider a smartphone user who is interested in participating in Nebula. Suppose they are willing to give up 5% of their battery throughout the course of a day and can spare 5 GB of storage on their phone, as they charge their phone overnight and have a 128 GB device. They usually keep their Bluetooth enabled, so when they join Nebula they do not incur the marginal cost of BLE listening. What is the maximum number of sensors they can interact with per day? In other words, what is the largest $f_{interact}$ possible without violating either their power or memory constraints?

Suppose the smartphone has a battery capacity of 4000 mAh $\approx$ 54720 J [84], [85]. Then the maximum power draw across a day would be $P_{max} = \frac{0.05 * 54720}{86400} = 31.67$ mW. Thus, $31.67\ mW \geq P_{collect} + P_{upload} = f_{interact} * (E_{interact} + E_{upload})$, so $f_{interact} \leq \frac{31.67 \times 10^{-3}}{E_{interact} + E_{upload}} = \frac{31.67 \times 10^{-3}}{0.0945 + 1.45 * 2} = 10.6$ mHz, so power constrains each mule to 915 sensor payloads per day.

Now let's consider the memory constraint. Suppose that each payload is around 1 kB of data. Then the spare 5 GB can fit 5 million payloads. If the mule stores all these payloads and uploads them once a month, they can pick

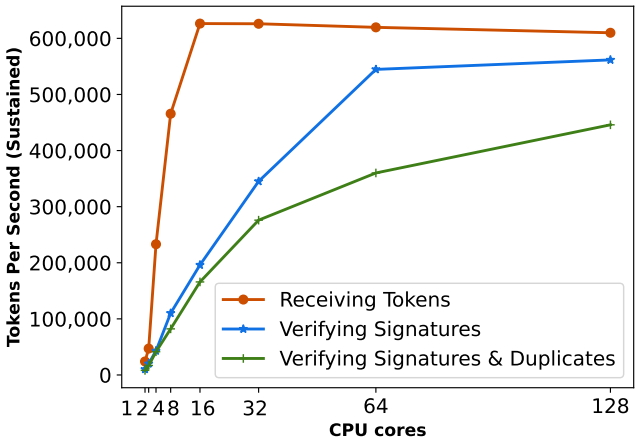

Figure 8: Number of tokens redeemed per second. With 128 cores, we can verify $445,900$ tokens, including filtering for duplicates, per second, or over 250 million tokens per dollar.

up $5,000,000/30 = 1666,666$ 1 kB payloads every day without running out of storage. On the other hand, if they upload each packet immediately after receiving it, they do not incur any memory costs for storing sensor data and can hold up to 50 million tokens, which are each 100 bytes long, before redeeming them at the end of the month. This would loosen their memory constraint to upper bound at $1,666,666$ arbitrarily-sized payloads every day.

It is clear that the energy constraint is far more restrictive than the memory constraint. Over the course of a month, this mule would upload at most $915 * 30 = 27450$ data payloads due to energy constraints, resulting in about 3 MB of storage used on the phone itself (assuming relatively frequent payload uploads). During the token redemption process, if the mule redeemed the tokens using a single HTTPS connection and uploading requests containing 100 tokens, it would take $2.4 + 275 * 0.8 = 223$ seconds $= 3.7$ minutes to redeem all the tokens. This redemption would consume $223 * 0.454 = 101$ J $\approx 0.18\%$ of the phone's battery.

8.4. Redemption

We discuss the performance of the Nebula platform provider when processing incoming tokens. When a mule redeems a set of tokens, the platform provider is responsible for verifying that the tokens are correctly signed and ensuring there no duplicate tokens were submitted, in order to determine how much to pay each mule. Our implementation showcases an efficient and cost-effective system, processing more than 445,000 tokens per second using a single node.

In Figure 8, we plot how the system scales with an increasing number of cores. We measure token processing rates under three different scenarios: (1) simply *receiving tokens* as fast as the HTTPS server can accept connections, (2) *verifying the signature* for each token, and (3) both verifying token signatures and ensuring that *duplicate* tokens are not redeemed twice. As CPU cores increase, we see substantial improvements in all three rates. Starting at around 16 cores, the rate at which our uvicorn HTTPS server can accept new connections becomes the the limiting factor. On a single core, the platform can receive $24,080$ tokens/sec, verify the blind signatures at $10,150$ tokens/sec, and check $8,190$ tokens/sec for duplicates. This scales up to 16 cores, to $626,360$, $196,140$, $165,830$ tokens/sec, respectively. With a full 128 cores, Nebula can fully process $445,900$ tokens per second. The GCP `n2-standard-128` instance on which we implemented our provider costs $6.2 an hour ($1.5 with Spot pricing). At $445,900$ tokens per second, Nebula can process 258.90 million tokens per dollar (1.06 billion tokens per dollar with Spot). Finally, mules are able to efficiently lodge complaints with the platform provider in response to misbehavior; our implementation requires 178 ms to validate a complaint based on an invalid token and 247 ms to validate a complaint involving an incomplete delivery.

Our implementation of the Nebula platform provider demonstrates high performance at low cost, while verifying token signatures and checking for duplicates. This allows for a system design that enables privacy-focused data backhaul that can accommodate a large number of participants and sensors with minimal constraints.

9. Conclusion

How can we massively extend network connectivity to the edge while protecting the privacy of the participants from a centralized platform? In this paper, we introduce Nebula, a decentralized architecture for privacy-preserving, general-purpose data backhaul. We experimentally show that our system incurs low energy and computational overheads, and we develop an analytical model to estimate real-world performance. Using Nebula, a smartphone anywhere in the world could backhaul almost a thousand data payloads a day consuming only 5% of its battery and 3MB of storage, without revealing its location to a central network server. For embedded sensing applications, our architecture vastly expands the scope of potential deployments while reducing the deployment cost.

Acknowledgments

We thank the anonymous reviewers and shepherd for their helpful feedback. We thank Michael Farley and Michele Orrù whose insightful comments helped shape this work, and the students in the Sky Computing security group for feedback that improved the presentation of this paper. This work is supported by gifts from AMD, Anyscale, Google, IBM, Intel, Microsoft, Mohamed Bin Zayed University of Artificial Intelligence, Samsung SDS, Uber, and VMware. This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the award number DE-EE0008220.

References

[1] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. N. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: An IoT platform for data-driven agriculture," in *Symposium on Networked Systems Design and Implementation*, 2017.

[2] J.-H. Huang, S. Amjad, and S. Mishra, "Cenwits: a sensor-based loosely coupled search and rescue system using witnesses," in *ACM Intl. Conf. on Embedded Networked Sensor Systems*, 2005.

[3] A. Pradeep, H. Javaid, R. R. Williams, A. Rault, D. R. Choffnes, S. L. Blond, and B. Ford, "Moby: A blackout-resistant anonymity network for mobile devices," *Proc. Priv. Enhancing Technol.*, 2022.

[4] N. Perry, B. Spang, S. Eskandarian, and D. Boneh, "Strong anonymity for mesh messaging," *ArXiv*, vol. abs/2207.04145, 2022.

[5] "Find My Network," https://developer.apple.com/find-my/, 2023.

[6] "How Tile Works," www.thetileapp.com/en-us/how-it-works, 2023.

[7] H. Louch, B. Davis, K. Voros, K. O'Toole, and S. Piper, "Innovation in bicycle and pedestrian counts," altago.com/wp-content/uploads/Innovative-Ped-and-Bike-Counts-White-Paper-Alta.pdf, 2016.

[8] "Amazon sidewalk privacy and security whitepaper," https://m.media-amazon.com/images/G/01/sidewalk/final_privacy_security_whitepaper.pdf, 2023.

[9] "Welcome to nRF Connect SDK - Amazon Sidewalk," https://nrfconnect.github.io/sdk-sidewalk/, 2023.

[10] T. Despres, S. Patil, A. Tan, J.-L. Watson, and P. Dutta, "Where the sidewalk ends: privacy of opportunistic backhaul," in *Proceedings of the 15th European Workshop on Systems Security*, 2022.

[11] M. Duckham and L. Kulik, "Location privacy and location-aware computing," in *Dynamic and mobile GIS*, 2006.

[12] B. Baron and M. Musolesi, "Where you go matters: a study on the privacy implications of continuous location tracking," *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2020.

[13] Y.-A. De Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Scientific reports*, vol. 3, no. 1, 2013.

[14] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously." *Proc. Priv. Enhancing Technol.*, 2018.

[15] H. Fan, H. Zhu, and D. Yuan, "People counting in elevator car based on computer vision," in *IOP Conference Series: Earth and Environmental Science*, 2019.

[16] M. Patil and V. N. Bhonge, "Wireless sensor network and rfid for smart parking system," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 4, pp. 188–192, 2013.

[17] M. Meyer, T. Farei-Campagna, A. Pasztor, R. D. Forno, T. Gsell, J. Faillettaz, A. Vieli, S. Weber, J. Beutel, and L. Thiele, "Event-triggered natural hazard monitoring with convolutional neural networks on the edge," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, 2019.

[18] "LoRa Alliance Site," www.lora-alliance.org, 2023.

[19] "LoRa README," https://lora.readthedocs.io/en/latest/, 2023.

[20] M. Lesund, "LTE-M vs NB-IoT field test," https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/ltem-vs-nbiot-field-test-how-distance-affects-power-consumption, 2022.

[21] "Nordic infocenter," https://infocenter.nordicsemi.com/, 2023.

[22] "ESP-IDK programming guide," https://docs.espressif.com/, 2023.

[23] "3GPP," https://www.3gpp.org/, 2023.

[24] P. Taylor, "Average cellular data price per gigabyte," https://www.statista.com/statistics/994913/average-cellular-data-price-per-gigabyte-in-the-us/, 2023.

[25] "Helium Explorer," https://explorer.helium.com/, 2023.

[26] A. Haleem, A. Allen, A. Thompson, M. Nijdam, and R. Garg, "Helium: A decentralized wireless network," 2018.

[27] D. Jagtap, A. Yen, H. Wu, A. Schulman, and P. Pannuto, "Federated infrastructure: usage, patterns, and insights from "the people's network"," *ACM Internet Measurement Conference*, 2021.

[28] A. Musaddiq, N. Maleki, F. Palma, D. Mozart, T. Olsson, M. Omareen, and F. Ahlgren, "Internet of things for wetland conservation using helium network: Experience and analysis," *International Conference on the Internet of Things*, 2022.

[29] B. Ghena, J. Adkins, L. Shangguan, K. Jamieson, P. A. Levis, and P. Dutta, "Challenge: Unlicensed lpwans are not yet the path to ubiquitous connectivity," *The 25th Annual International Conference on Mobile Computing and Networking*, 2019.

[30] A. Greenberg, "The clever cryptography behind Apple's 'Find My' feature," https://www.wired.com/story/apple-find-my-cryptography-bluetooth/, 2019.

[31] A. Heinrich, M. Stute, and M. Hollick, "Openhaystack: a framework for tracking personal bluetooth devices via apple's massive find my network," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021.

[32] A. Bellon, A. Yen, and P. Pannuto, "Tagalong: Free, wide-area data-muling and services," in *Proceedings of the 24th International Workshop on Mobile Computing Systems and Applications*, 2023.

[33] J. Chase, "Amazon sidewalk will share your internet with strangers. it's not as scary as it sounds." https://www.nytimes.com/wirecutter/blog/amazon-sidewalk-review/, 2021.

[34] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. I. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet," in *ASPLOS X*, 2002.

[35] A. M. Mainwaring, D. E. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *ACM International Conference on Wireless Sensor Networks and Applications*, 2002.

[36] J. C. Haartsen, "Bluetooth-ad-hoc networking in an uncoordinated environment," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2001.

[37] J. M. Rabaey, M. J. Ammer, J. L. da Silva, D. Patel, and S. Roundy, "Picoradio supports ad hoc ultra-low power wireless networking," *Computer*, vol. 33, 2000.

[38] M. S. Corson and J. P. Macker, "Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations," *RFC*, 1999.

[39] S. Lan, Z. Wang, J. Mamish, J. D. Hester, and Q. Zhu, "Adasens: Adaptive environment monitoring by coordinating intermittently-powered sensors," *Asia and South Pacific Design Automation Conference*, 2022.

[40] T. A. Wild, L. van Schalkwyk, P. Viljoen, G. Heine, N. Richter, B. Vorneweg, J. C. Koblitz, D. K. N. Dechmann, W. Rogers, J. Partecke, N. Linek, T. Volkmer, T. Gregersen, R. W. Havmøller, K. Morelle, A. Daim, M. Wiesner, K. Wolter, W. Fiedler, R. W. Kays, V. O. Ezenwa, M. Meboldt, and M. Wikelski, "A multi-species evaluation of digital wildlife monitoring using the sigfox iot network," *Animal Biotelemetry*, vol. 11, 2023.

[41] V. Venkataramanan, G. M. Kavitha, M. R. Joel, and J. Lenin, "Forest fire detection and temperature monitoring alert using iot and machine learning algorithm," *International Conference on Smart Systems and Inventive Technology*, 2023.

[42] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "Applications on the signpost platform for city-scale sensing: demo abstract," in *International Symposium on Information Processing in Sensor Networks*, 2018.

[43] "U.K. bars trash cans from tracking people with wi-fi," *CBS News*, 2013.

[44] S. Ali, T. Osman, M. Mannan, and A. Youssef, "On privacy risks of public wifi captive portals," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, 2019, pp. 80–98.

[45] R. Want, B. N. Schilit, and S. Jenson, "Enabling the internet of things," *Computer*, vol. 48, pp. 28–35, 2015.

[46] S. Jenson, R. Want, B. N. Schilit, and R. Kravets, "Building an on-ramp for the internet of things," *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*, 2015.

[47] "Swarm starlink," https://swarm.space/, 2023.

[48] F. Michel, M. Trevisan, D. Giordano, and O. Bonaventure, "A first look at starlink performance," *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022.

[49] F. C. Uyeda, M. Alvidrez, E. Kline, B. Petrini, B. J. Barritt, D. Mandle, and A. C. Alexander, "SDN in the stratosphere: loon's aerospace mesh network," *ACM SIGCOMM*, 2022.

[50] Apple/Google, "Exposure notification - bluetooth specification," shttps://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ExposureNotification-BluetoothSpecificationv1.2.pdf?1, 2020.

[51] M. Jawurek, F. Kerschbaum, and G. Danezis, "Privacy technologies for smart grids - a survey of options," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:15815464

[52] R. A. Popa, H. Balakrishnan, and A. J. Blumberg, "VPriv: protecting

privacy in location-based vehicular services," in *USENIX Security Symposium*, 2009.
[53] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li, "Privacy and accountability for location-based aggregate statistics," in *ACM Conference on Computer and Communications Security*, 2011.
[54] D. Chaum, "The Dining Cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, 1988.
[55] M. Lentz, V. Erdélyi, P. Aditya, E. Shi, P. Druschel, and B. Bhattacharjee, "Sddr: Light-weight, secure mobile encounters," in *USENIX Security Symposium*, 2014.
[56] J. Manweiler, R. Scudellari, and L. P. Cox, "Smile: encounter-based trust for mobile social services," in *ACM Conference on Computer and Communications Security*, 2009.
[57] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy," in *USENIX Security*, 2021.
[58] R. Cheng, W. Scott, E. Masserova, I. Zhang, V. Goyal, T. Anderson, A. Krishnamurthy, and B. Parno, "Talek: Private group messaging with hidden access patterns," in *Annual Computer Security Applications Conference*, 2020.
[59] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *SOSP*, 2015.
[60] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.
[61] A. H. Kwon, D. Lazar, S. Devadas, and B. Ford, "Riffle: An efficient communication system with strong anonymity," 2016.
[62] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE Symposium on Security and Privacy*, 2015.
[63] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, "Anonymous tokens with private metadata bit," in *CRYPTO*, 2020.
[64] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.
[65] L. Yang and F. Li, "mTor: A multipath tor routing beyond bandwidth throttling," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015.
[66] M. Rennhard and B. Plattner, "Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, 2002.
[67] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001.
[68] H. Givehchian, N. Bhaskar, E. R. Herrera, H. R. L. Soto, C. Dameff, D. Bharadia, and A. Schulman, "Evaluating physical-layer ble location tracking attacks on mobile devices," *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.
[69] J. K. Becker, D. Li, and D. Starobinski, "Tracking anonymized bluetooth devices," *PET*, 2019.
[70] B. Ledvina, Z. Eddinger, B. Detwiler, and S. P. Polatkan, "Detecting Unwanted Location Trackers," Internet-Draft draft-detecting-unwanted-location-trackers-00, 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-detecting-unwanted-location-trackers/00/
[71] A. Heinrich, N. Bittner, and M. Hollick, "AirGuard - protecting android users from stalking attacks by apple find my devices," *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2022.
[72] J. Briggs and C. Geeng, "Ble-doubt: Smartphone-based detection of malicious bluetooth trackers," *2022 IEEE Security and Privacy Workshops (SPW)*, 2022.
[73] S. Kandula, D. Katabi, M. Jacob, and A. Berger, "Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds," in *PNSDI*.
[74] K. Bhardwaj, J. C. Miranda, and A. Gavrilovska, "Towards IoT-DDoS prevention using edge computing," in {*USENIX*} *Workshop on Hot Topics in Edge Computing*, 2018.
[75] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer dos defense against multimillion-node botnets," in *ACM SIGCOMM*, 2008.
[76] "Certificate Transparency," https://certificate.transparency.dev/, last visited on December, 2023.
[77] N. Jackson, J. Adkins, and P. Dutta, "Capacity over capacitance for reliable energy harvesting sensors," in *The 18th International Conference on Information Processing in Sensor Networks*, 2019.
[78] "Anonymous tokens: Efficient anonymous tokens with private metadata bit," https://github.com/mmaker/anonymous-tokens, 2023.
[79] "BLE security," https://www.bluetooth.com/bluetooth-resources/le-security-study-guide/, 2023.
[80] N. Klugman, V. Jacome, M. Clark, M. Podolsky, P. Pannuto, N. Jackson, A. S. Nassor, C. Wolfram, D. Callaway, J. Taneja *et al.*, "Experience: Android resists liberation from its primary use case," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018.
[81] "Android studio — power profiler," https://developer.android.com/studio/profile/power-profiler, 2023.
[82] Sparkfun, "Coin cell battery - 20mm (CR2032)," https://www.sparkfun.com/products/338, 2023.
[83] Microbattery, "Battery bios: Everything you need to know about the aa battery," https://www.microbattery.com/blog/post/battery-bios:-everything-you-need-to-know-about-the-aa-battery/.
[84] A. Authority, "You told us: This is the battery capacity of most of your smartphones right now," https://www.androidauthority.com/smartphone-battery-size-poll-results-1221015/, 2021.
[85] Deloitte, "Smartphone batteries: better but no breakthrough," https://www2.deloitte.com/content/dam/Deloitte/global/Documents/Technology-Media-Telecommunications/gx-tmt-pred15-smartphone-batteries.pdf, 2015.

Appendix A. Soundness

We prove below a group of closely-related properties related to the soundness of the core Nebula protocol: payload delivery and complaints.

Claim 1 (Nebula valid token exchange)**.** *Assuming an existentially unforgeable signature scheme, in the Nebula system, any mule M that delivers a payload to an honest AS using Algorithm 2 will receive token t only if* $\mathsf{Verify}(P_{hash}, \sigma_{hash}, pk_s)$ *succeeds in Algorithm 2 step 3.*

Proof Sketch. If the check at step 3 fails, the AS aborts and returns $\perp$ instead of a valid token. Soundness follows from the *existential unforgeability* of the signature scheme, because the signature σ_{hash} on P_{hash} could only have been generated by the sensor who knows pk_s. □

Claim 2 (Nebula duplicate payload upload)**.** *Assuming an existentially unforgeable signature scheme, a semantically secure encryption scheme, and a collision-resistant hash function, in the Nebula system, any mule M that delivers a valid payload twice to an honest AS using Algorithm 2 (i.e.* $\mathsf{Verify}(P_{hash}, \sigma_{hash}, pk_s)$ *succeeds in Algorithm 2 step 3) will receive at most one token t.*

Proof Sketch. On the first and second deliveries, if the payload d does not match P_{hash}, M will not receive any tokens per Claim 1. If M can present a valid P_{hash}, σ_{hash}, and d to the AS, on the first delivery, AS can either abort the protocol, yielding no token, or send one valid token t in P_{token}. AS stores the payload hash $H(d)$, such that on the second delivery, the payload hash will match and abort the protocol. This follows from the *deterministic* property of the hash function, and results in at most one token. Note that

while an encrypted token is present in P_{pre}, the *semantic security* of the encryption scheme ensures that only AS or P could decrypt the ciphertext to reveal a valid token. M could attempt to file a complaint with the provider using Algorithm 4 in a bid to retrieve another token using the (P_{pre}, σ_{pre}) it received in Algorithm 2 step 5 and either d or (P_{token}, σ_{token}) from Algorithm 2 step 10. In the former case, the provider would simply invalidate the token returned by AS in Algorithm 4 step 3 before generating the new token, and in the latter case, the provider would abort in step 4 as the original token is still the one valid token. □

Claim 3 (Nebula token guarantee)**.** *Assuming an Unlinkable Token scheme with unlinkability and one-more-token security guarantees, an existentially unforgeable signature scheme, a collision-resistant hash function, and a semantically secure symmetric encryption scheme, in the Nebula system, for any honest mule M, for any application server AS, if M validates P_{pre} in Algorithm 2 step 6, M will either receive a token t that can be successfully redeemed in Algorithm 3 or convince the provider of AS misbehavior.*

Proof Sketch. After receiving a P_{pre} message from the AS with a valid signature σ_{pre}, there are four cases:
Case 1. Upon M sending d to AS, AS does not respond with P_{token} and σ_{token}; the AS is attempting to steal the data without rewarding M with a token. M saves a complaint record as described in Algorithm 2 and, at the end of the epoch, lodges a complaint using Algorithm 4. P verifies that P_{pre} is correct, which relies on the *existential unforgeability* of the signature scheme, and M sends the payload d to P which matches the hash in P_{pre}. Given the *collision-resistant* property of the hash function, this is the payload that the AS originally committed to receiving, so P forwards the payload to AS and signs a new token for M. M can verify that this token was signed with the correct key, following from the Unlinkable Token scheme's *unlinkability* guarantee, resulting in a valid redeemable token t^*.
Case 2. AS responds with P_{token} and σ_{token}, but the token t contained in P_{token} is flagged as invalid by P in Algorithm 3; the AS gave M an invalid token. M saves a complaint record and lodges a complaint at the end of the epoch using Algorithm 4. P verifies σ_{token}, which relies on the *existential unforgeability* of the signature scheme, and that P_{token}'s token cannot be verified using the Unlinkable Token scheme. This implicates AS in providing an invalid token, because AS checks that tokens are correctly signed by P at purchase time, which follows from the *unlinkability* guarantee. P signs a new token for M, which M can likewise verify, resulting in a valid redeemable token t^*.
Case 3. AS responds with a P_{token} and σ_{token} but the token t in P_{token} is a duplicate of a token already redeemed. M and the mule that redeemed t earlier can lodge a complaint at the end of the epoch, with P verifying σ_{token} as above in Case 2. As the AS gave M an already-used token, the different P_{token} payloads in each complaint will contain the same t, convincing P that the signing AS(es) are misbehaving.
Case 4. AS responds with a P_{token} and σ_{token} containing a valid redeemable token t. □

Appendix B. Privacy

In this appendix, we define Nebula's privacy guarantee in the simulation paradigm [67] (note that Nebula's privacy could also be defined and proved in a computational indistinguishability framework depending on the guarantees of the underlying building blocks). We give an overview of the proof and connect it to the informal privacy guarantees in Section 4, then present the structure of the simulation and a proof sketch for indistinguishability.

The real and ideal world are depicted in Figure 9. In the real world, honest application servers and mules (AS_H and M_H, respectively) interact with a malicious provider, application servers, and mules (P, AS_C, and M_C) controlled by $\mathcal{A}$. In the ideal world, the honest parties are represented by an uncorruptible trusted party $\mathcal{F}$ called an ideal functionality; on every operation, $\mathcal{F}$ provides a simulator party $\mathcal{S}$ with a well-defined subset of information about the operation. This subset defines what information Nebula leaks to $\mathcal{A}$ and provides a clear definition of privacy in our setting. $\mathcal{S}$ then interacts with $\mathcal{A}$ to complete the operation. $\mathcal{S}$, however, cannot perform the operation exactly as an honest party because it does not know all operation inputs, only the subset it was given by $\mathcal{F}$. $\mathcal{S}$ must simulate the operation such that what $\mathcal{A}$ sees is computationally indistinguishable from what it would see in the real world. The existence of $\mathcal{S}$ that can properly simulate the Nebula protocol would show that Nebula reveals no more to $\mathcal{A}$ than what $\mathcal{F}$ gives $\mathcal{S}$ on each operation.

B.1. Limitations

For simplicity and to enable us to focus on the core behavior of Nebula, we do not model all aspects of the complex real system, as follows. First, in practice, epochs overlap to avoid losing data from payload pickups near the end of the epoch: we model one epoch at a time in sequence. Second, we don't directly model network connections, including network information that might be leaked when honest parties connect to corrupted participants, because parties can hide this information using complementary systems like Tor Section 4.3. We also do not model messages dropping/becoming corrupted during transport. Third, we don't model timing or concurrent operations (although our implementation of Nebula in Section 7 handles concurrent redemption), and assume that we process one operation at a time. Fourth, we assume the sensors and payment mechanism are external, as they have little bearing on performing the protocol: sensors simply generate the inputs to the mules, which we model as appearing directly in X, and we assume the provider can charge for executing **purchase_tokens** operations. Fifth, application servers and mules may be malicious, but we restrict $\mathcal{A}$ to static compromise – honest parties do not later become malicious. Finally, our definition here captures the privacy that mules have in Nebula, and not the confidentiality of the payload from the sensors (which is taken care off independently via end-to-end encryption).

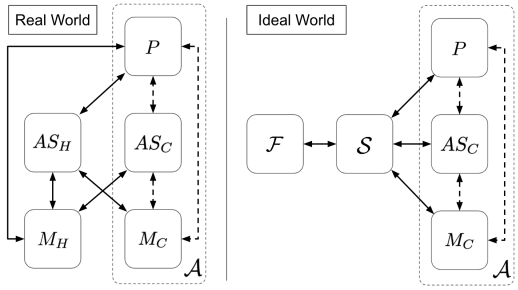

Figure 9: Overview of the real and ideal worlds.

B.2. Real World

In the real world (Figure 9), honest mules M_H and application servers AS_H (the honest parties) interact directly with the mules, application servers, and provider (M_C, AS_C, and P, respectively) corrupted by $\mathcal{A}$ according to the Nebula protocol described in Section 5. The honest parties are given a sequence of operations $X = \{x_1, ..., x_n\}$ to execute, where each operation x_i is one of the following operations: **purchase_tokens**(id$_{AS}$, n_t), **deliver**(id$_M$, id$_{AS}$, (d, P_{hash}, σ_{hash})), **redeem**(id$_M$, T), and **new_epoch**(n_c, T_{dup}).

Valid sequences X have a specific structure, which models the flow of the Nebula protocol over the course of each epoch. Specifically, for any X of length n, X begins and ends with a new epoch – x_1 and x_n are always **new_epoch** – and normal operations and new epochs are interleaved in the rest of the sequence – every x_i for $1 < i < n$ is one of **purchase_tokens**, **deliver**, **redeem**, or **new_epoch**.

In each X, the mules and application servers that should execute the operation are identified by id$_M$ and id$_{AS}$, respectively. n_t is the number of payloads an application server should prepay for delivery (e.g. number of tokens an application server should purchase), n_c the number of complaints a mule can make in that epoch, T is a list of delivery tokens a particular mule attempts to redeem with the provider, d is the sensor data to deliver, and T_{dup} is a list of the mule's submitted tokens that saw duplicate submissions.

Operations that do not include an honest party are not included in X because $\mathcal{A}$ already has perfect visibility on the operation, and need not follow the Nebula protocol at all. However, operations that are only visible to honest parties *are* included in X, as this ensures that our proof demonstrates that interactions been honest parties do not affect participant privacy.

B.3. Ideal World

In the ideal world (Figure 9), we define an ideal functionality $\mathcal{F}$ that captures Nebula's privacy guarantees. $\mathcal{F}$ receives sequence X as input and executes the operations one at a time in the order provided. To perform each individual operation, $\mathcal{F}$ interacts with the simulator $\mathcal{S}$ as specified below, with communication from $\mathcal{F}$ to $\mathcal{S}$ underlined. $\mathcal{S}$, in turn, simulates the Nebula protocol toward $\mathcal{A}$ based on the subset of each call with which it was provided.

For **new_epoch**(n_c, T_{dup}), $\mathcal{F}$ marks the next epoch as active, initializes each honest mule's complaint counter to n_c, and sends **new_epoch**(n_c, T_{dup}) to $\mathcal{S}$.

For **complain**(c) from $\mathcal{S}$, $\mathcal{F}$ finds the mule corresponding to c, and if its complaint counter is greater than 0, sends $\top$ to $\mathcal{S}$ and decrements the counter. Else, $\mathcal{F}$ sends $\bot$.

For **purchase_tokens**(id$_{AS}$, n_t), $\mathcal{F}$ adds n_t to id$_{AS}$'s purchased token counter for this epoch, and sends **purchase_tokens**(id$_{AS}$, n$_t$) to $\mathcal{S}$.

For **deliver**(id$_M$, id$_{AS}$, (d, P_{hash}, σ_{hash})), there are 4 cases: (1) If both id$_M$ corresponds to an honest mule (id$_M$ in M_H) and id$_{AS}$ corresponds to an honest application server (id$_{AS}$ in AS_H), check for a duplicate payload hash $H(d)$ and: decrement id$_{AS}$'s purchased token counter if is greater than 0 and increment id$_M$'s delivered payload counter by 1 and add $H(d)$ to id$_{AS}$'s received payload list, else do nothing. $\mathcal{F}$ does not send anything to $\mathcal{S}$. (2) If id$_M$ corresponds to an honest mule but id$_{AS}$ corresponds to a malicious application server (id$_{AS}$ in AS_C): increment id$_M$'s delivered payload counter by 1 and send $\mathcal{S}$ the operation **deliver**(_, id$_{AS}$, (d, P_{hash}, σ_{hash})), where the "_" notation indicates that the first argument (in this case, id$_M$) is not sent to the simulator. (3) If id$_M$ corresponds to a malicious mule (id$_M$ in M_C) but id$_{AS}$ corresponds to an honest application server, check for a duplicate payload hash $H(d)$ and: decrement id$_{AS}$'s purchased token counter if greater than 0, add $H(d)$ to id$_{AS}$'s received payload list, and send $\mathcal{S}$ $\top$ to denote success, else send $\mathcal{S}$ $\bot$ to denote failure. (4) If id$_M$ corresponds to a malicious mule and id$_{AS}$ is honest but d is missing (only P_{hash} is provided by $\mathcal{S}$), check for a duplicate payload hash $H(d)$ and id$_{AS}$'s purchased token counter is greater than 0 and send $\mathcal{S}$ $\top$ to denote no duplicate or $\bot$ otherwise.

For **redeem**(id$_M$, T), $\mathcal{F}$ calculates the redemption size n as $\min(|T|, \text{id}_M \text{ delivered payload counter})$, decrements id$_M$'s delivered payload counter by n, and sends $\mathcal{S}$ the operation **redeem**(id$_M$, n).

B.3.1. Relation to informal properties in Section 4.2

$\mathcal{F}$ sends a set of information to $\mathcal{S}$ above that correspond to the informal privacy properties listed in the main text. Specifically, $\mathcal{S}$ sees (1) each **purchase_tokens** operation, allowing $\mathcal{A}$ to see how many payload deliveries each AS prepurchases in an epoch, (2) the count of tokens in each **redeem** operation, which leaks to $\mathcal{A}$ how many tokens overall a mule redeems in each epoch, and (3) each **complain** operation without mule identifiers, leaking to $\mathcal{A}$ each complaint registered against each AS. In addition, $\mathcal{A}$ sees when new epochs begin, duplicate token uploads, the payloads delivered to malicious ASes, and if uploads succeed from malicious mules.

B.4. Proof

We prove Theorem 1 by constructing a simulator $\mathcal{S}$ for Nebula that, given the information provided by $\mathcal{F}$ on each operation, interacts with $\mathcal{A}$ so that it cannot distinguish the real world from the ideal world. For readability, we say "$\mathcal{A}$ cannot distinguish the real world from the ideal world" in this section to mean

the cryptographic equivalence stated in Definition 1 applied to π_{Nebula}: $\exists\mathcal{S}\forall\mathcal{A}\forall X\{\text{REAL}_{\pi_{Nebula},\mathcal{A}(z),X}(1^\lambda)\}_\lambda \overset{c}{\equiv} \{\text{IDEAL}_{\mathcal{S},\mathcal{A}(z),X}(1^\lambda)\}_\lambda$.

$\mathcal{S}$ interacts with $\mathcal{A}$ on an operation-by-operation basis, just as honest mules or application servers would in the real world. $\mathcal{F}$ is designed to avoid giving $\mathcal{S}$ mule identifiers when delivering payloads (or complaining about payload delivery), so $\mathcal{S}$'s main role is to act as (1) a mule performing all honest backhaul operations in the system and (2) an honest application server managing malicious mules. Nebula is designed such that the malicious provider, application servers, and mules cannot distinguish this case from individual mules each providing a subset of the backhaul service.

We use a unlinkable token (UT) protocol as a building block in Nebula. Let $\mathcal{S}_{\text{UT}}$ be a simulator for a UT protocol with the *unlinkability* and *one-more-token* security guarantees as described in Section 2.3. We instantiate $\mathcal{S}_{\text{UT}}$ with the PrivacyPass protocol [14], [63]; given that Davidson et al. [14] do not formally provide an existing simulator for PrivacyPass, and it is out-of-scope in this work to create it, we assume a simulator with an interface as defined below.

Definition 2 ((Informal) Simulator for Unlinkable Tokens)**.** *$\mathcal{S}_{UT}$ has the following interface:*

- *$\mathcal{S}_{UT}$.KeyGen() simulates generating a new keypair and publishing its public parameters to $\mathcal{A}$*
- *$\mathcal{S}_{UT}$.Sign$_k(t')$ simulates signing a given token value t' under some key pair k*
- *$\mathcal{S}_{UT}$.Redeem$_k(t)$ simulates redeeming a signed token t under some key pair k*

B.4.1. Description of $\mathcal{S}$

In the setup phase, $\mathcal{A}$ simulates $\mathcal{S}_{\text{UT}}$.KeyGen() twice to generate epoch delivery and complaint keys k_c and k_d, and reveals the public params to $\mathcal{S}$. $\mathcal{A}$ generates a separate symmetric key k_i for each id$_{AS}$ in AS_H and sends to $\mathcal{S}$, and $\mathcal{S}$ generates public-private keypairs for each AS in AS_H and a keypair for every sensor s controlled by an honest AS; $\mathcal{A}$ gives the public keypairs for each AS in AS_C to $\mathcal{S}$.

For **start_epoch**, $\mathcal{S}$ simulates $\mathcal{S}_{\text{UT}}$.Sign$_{k_c}(t)$ for every t in a random vector of n$_c$ blinded tokens and appends each token list to the epoch master Mule complaint token list. $\mathcal{S}$ adds the complaint records the tokens in T_{dup} to its complain list. Then, for every record c in the complaint list, $\mathcal{S}$ sends **complain**(c) to $\mathcal{F}$. If $\mathcal{F}$ returns $\top$, $\mathcal{S}$ pops a token off of last epoch's master Mule complaint token list and sends it along with c to $\mathcal{A}$. If c contains P_{token}, $\mathcal{S}$ sends $(P_{token}, \sigma_{token})$, else d to $\mathcal{A}$. If $\mathcal{A}$ does not reply with $\perp$, $\mathcal{S}$ simulates $\mathcal{S}_{\text{UT}}$.Sign$_{k_d}(t)$ on a new token t with $\mathcal{A}$ and appends it to the new epoch Mule delivery token list.

For **purchase_tokens**, $\mathcal{S}$ simulates $\mathcal{S}_{\text{UT}}$.Sign$_{k_d}(t)$ for every t in a random vector of n$_t$ blinded tokens. $\mathcal{S}$ appends the tokens to the epoch master AS token list.

For **deliver**, there are two cases. First, if $\mathcal{S}$ receives the operation from $\mathcal{F}$ to deliver a payload to a malicious id$_{AS}$: $\mathcal{S}$ sends the AS (P_{hash}, σ_{hash}), verifies the resulting σ_{pre} using the right sk_{as}, sends d to $\mathcal{A}$, verifies the resulting σ_{token} and saves the output token in the epoch master Mule delivery token list and a complaint record. Second, if $\mathcal{S}$ receives the operation from $\mathcal{A}$ to deliver a payload to an honest id$_{AS}$: $\mathcal{S}$ checks σ_{hash} using the correct sensor secret key and sends P_{hash} to $\mathcal{F}$ to check for duplicate. On success, $\mathcal{S}$ picks a random r, pops a token t from the master AS token list, and uses its keys to generate P_{pre} and sign σ_{pre} for $\mathcal{A}$. When $\mathcal{A}$ sends d, $\mathcal{S}$ checks that the payload hash matches and generates and sends P_{token} and σ_{token}.

For **redeem**, $\mathcal{S}$ pops n tokens from the epoch's master Mule delivery token list and sends the list to $\mathcal{A}$. On response with the invalid token list, $\mathcal{S}$ adds the matching complaint records to its complaint list.

B.4.2. Proof Sketch for Indistinguishability

In this subsection, we sketch a proof for why $\mathcal{A}$ cannot distinguish the real world from the ideal world for each operation executed by the simulator $\mathcal{S}$.

Proof Sketch. For **start_epoch**, $\mathcal{S}$ uses $\mathcal{S}_{\text{UT}}$ to simulate the complaint token generation. Since this simulation is, by definition, indistinguishable, and $\mathcal{S}$ knows to sign $|M_H|$ vectors of n_c elements at each epoch, $\mathcal{A}$ sees indistinguishable signing requests from the real world. This follows from the computational indistinguishability of the random blinded tokens $\mathcal{S}$ submits to $\mathcal{A}$. When handling complaints, $\mathcal{S}$ interacts with $\mathcal{F}$ to ensure it submits at most n_c' complaints per mule. Since $\mathcal{S}$ only submits complaints from its honest mules about malicious ASes, it can retain the necessary complaint record c from a previous **deliver** operation and perfectly replay the messages to $\mathcal{A}$ to submit the complaint as in the real world. Finally, $\mathcal{S}_{\text{UT}}$ indistinguishably simulates the signing of a new token.

For **purchase_tokens**, $\mathcal{S}$ again uses $\mathcal{S}_{\text{UT}}$ to simulate the delivery token generation. Since this simulation is indistinguishable, and $\mathcal{S}$ knows how many tokens to sign (n_t), it can generate its own random token values, and $\mathcal{A}$ will see an indistinguishable signing request from the real world, following from the computationally indistinguishable random blinded tokens $\mathcal{S}$ submits to $\mathcal{A}$.

For **deliver**, in the first case, $\mathcal{S}$ is given the payload to deliver (d, P_{hash}, σ_{hash}) and so can exactly replicate the payloads to $\mathcal{A}$ for the delivery protocol as if in the real world. In the second case, $\mathcal{S}$ uses $\mathcal{F}$ to check for duplicates, matching exactly the real-world abort behavior in case of duplicate uploads. $\mathcal{S}$ picks a random token indistinguishable from what the honest AS would choose in the real world and takes the first available delivery token to return to $\mathcal{A}$. These signed tokens, following from the unlinkable token scheme's *unlinkability* guarantee [14], are computationally indistinguishable to $\mathcal{A}$, so it doesn't matter which particular token $\mathcal{S}$ chooses from the epoch's master AS token list; thus, P_{token} and σ_{token} are indistinguishable to $\mathcal{A}$.

Finally, for **redeem**, $\mathcal{S}$ knows exactly how many tokens to submit, and, similarly to the case above, the choice of tokens does not matter (any set of n tokens in the epoch's master Mule delivery token list is valid), as each signed token is computationally indistinguishable in $\mathcal{A}$'s view. Thus, $\mathcal{A}$ sees an indistinguishable token list from $\mathcal{S}$. □

Appendix C.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

The authors tackle the problem of supporting data backhaul for IoT devices. In contrast to much prior work and existing deployed systems, the authors look to not only support a privacy-preserving system for the ”mules” that help transport the data from the IoT devices, but also incorporate incentives via micropayments with support for addressing abuse complaints. The authors develop the Nebula architecture that meets all of these goals, with formal reasoning about the security properties that it provides. The authors develop a real-world implementation of Nebula to examine the energy efficiency and cloud platform performance of their solution, along with an analytical model to reason about energy efficiency in various parameterized settings.

C.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) Provides a Valuable Step Forward in an Established Field: Given the roll-out of systems like Sidewalk and FindMy, reviewers appreciated a privacy-focused protocol design. Elements like incentives and abuse complaints added further depth to the contribution.

C.4. Noteworthy Concerns

1) Some reviewers had concerns about the choice to have mules connect to applications directly. This choice required additional steps by a mule to ensure anonymity (e.g., Tor), and it’s not clear that an application would be more trustworthy than the service provider.

Appendix D.
Response to the Meta-Review

While mules in Nebula must use additional anonymity mechanisms to connect directly to application servers (e.g. Tor) as noted in reviewers’ concern above, Nebula does not place any more trust in the application than it is given in a centralized setting. We note in Section 4.2 that Nebula’s contribution is to remove the provider entirely from the data path, preventing it from aggregating data from all mules across all applications, and that applications receive the same data payloads in Nebula that they would in any other backhaul network (e.g. Sidewalk).