Twofer: Ambiguous Transmissions for Low-Latency Sensor Networks Facing Noise, Privacy and Loss

Jonathan Oostvogels
DistriNet, KU Leuven
3001 Leuven, Belgium
jonathan.oostvogels@kuleuven.be

Sam Michiels
DistriNet, KU Leuven
3001 Leuven, Belgium
sam.michiels@kuleuven.be

Danny Hughes
DistriNet, KU Leuven
3001 Leuven, Belgium
danny.hughes@kuleuven.be

ABSTRACT

Today's wireless sensor networks focus on achieving reliable data transfer over a lossy medium at the expense of latency. However, sensor data are often noisy and thus only lossily characterise real-world phenomena, rendering their exact transfer wasteful. Furthermore, many next-generation privacy-sensitive applications, such as smart grid control, real-time distributed object tracking, and inter-vehicle federated learning face latency and traffic bottlenecks due to the sheer amount of data collection required to overcome noise. We tackle this problem by introducing Twofer, a communication approach which reduces latency and traffic in high-noise or high-privacy settings by abandoning the focus on reliable networking. Twofer empowers developers to tune networks for latency-bound rather than reliability-bound performance; the system coordinates *ambiguous* transmissions, which are used to estimate the network-wide distribution of data, rather than to reliably communicate exact data from individual nodes. Twofer's full-stack design maintains black-box compatibility with existing application code, but advocates for, and shows the value of, uncommon physical-layer features such as *symbol-synchronous* communication. The system is therefore implemented and evaluated on a prototype low-latency wireless mesh network called Zero-Wire. Experiments using state-of-the-art local differential privacy protocols show 25–75% latency reductions relative to conventional approaches. The results are also future-proof, with performance advantages increasing with the strength of the privacy guarantees that are offered.

KEYWORDS

bus network, synchronous transmission, symbol-synchronous, network stack, multiple-access channel, local differential privacy, over-the-air, quantisation, dithering, semantic communication, local differential privacy

1 INTRODUCTION

Distributed sensor systems value the data they consume in a profoundly different way than user-centric software systems. The latter are primarily concerned with exact data transfer to or from a non-redundant device. Examples include the transfer of passwords, photos, or web pages via a user's smartphone. In contrast, for distributed sensor applications, exact, reliable, and point-to-point data transfer is less valuable. Applications must inevitably tolerate somewhat inaccurate or partial information due to sensor noise or link failures. Noise and loss are therefore a twofer: applications must account for their impact, but networks can to some extent be freed from the burden of exact and reliable transfer.

Today's wireless sensor network technologies do not reflect applications' differing attitudes towards the data they consume. On the contrary, technologies such as IEEE 802.15.4 [27], Bluetooth [24], or LoRa [43] go to great lengths to ensure or improve the reliability of data transfer. The cost of this reliability focus is invariably traffic and latency overhead, for example, due to redundancy in modulation [58], link-layer re-transmissions [27], and carefully orchestrated medium access control strategies [27]. Noise in data itself adds further latency, as key applications such as smart grid control [12], real-time distributed object tracking [18], or inter-vehicle federated learning [61] must aggregate data from many densely deployed nodes to overcome the noise inherent in individual measurements. The current quest for sensor platforms that provide privacy guarantees for this class of applications [14, 22, 32, 41] adds another layer of latency, as contemporary notions of sensor privacy, e.g. local differential privacy [14, 62], essentially add noise prior to transmission and thus increase the amount of data that needs to be gathered to ensure a given performance level [33, 60]. High-privacy (and hence high-noise) wireless sensor systems therefore face prohibitive network-induced latency [20, 50].

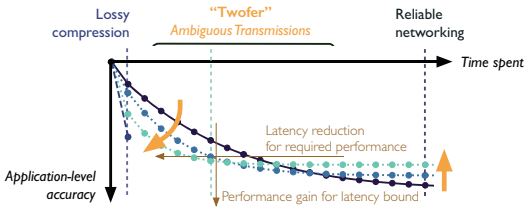

Figure 1: Twofer tunes latency-accuracy trade-offs for latency-bound rather than asymptotic performance.

This paper is the first to address the latency issues faced by noisy sensor networks by tuning PHYsical-layer (PHY) semantics to the available latency budget. In particular, it introduces *ambiguous* transmissions, i.e. concurrent transmissions that communicate frequency estimates for sets of data values, rather than exact data values from a single node. As illustrated in Figure 1, this approach trades performance that could be achieved if latency budgets were arbitrarily lenient for performance that can effectively be achieved under latency constraints. The proposed system, called Twofer, thus occupies a middle ground between on-node compression systems and reliable network stacks: it lossily estimates a carefully parametrised set of histograms that describe distributed nodes' data, rather than reliably collecting that data. At the physical layer, Twofer exploits on-the-air quantisation: multiple nodes transmit concurrently, and quantisation takes place through interactions between concurrent transmissions. Twofer is therefore prototyped on the Zero-Wire hardware platform [47], which provides PHY support. Empirical validation of Twofer shows 25–75% traffic reductions for data collection under currently advocated privacy regimes. These results demonstrate that conventional low-level

979-8-3503-6201-5/24/$31.00 ©2024 IEEE
DOI 10.1109/IPSN61024.2024.00022

network design considerations such as error correction, eventual reliability, and asymptotic performance no longer make sense for emerging application classes.

The remainder of this paper details its contribution. Section 2 provides background on noise, privacy, and the features provided by Zero-Wire. Section 3 offers an overview of how Twofer collects distributed data. Section 4 details the system's operation. Section 5 highlights implementation aspects. Section 6 empirically evaluates Twofer and discusses the results. Section 7 elaborates on related work. Section 8 concludes the paper.

2 BACKGROUND

2.1 Noise and privacy

Noise is what makes useful signals difficult to infer. "Low-noise" communication therefore makes some concept of interest X easy to accurately guess, given a message Y: $Pr(X = x_i|Y)$ concentrates as much as possible probability on as few as possible values x_i for X [51]. Low-noise modulation thus enables the reliable recovery of transmitted symbols (X) from samples seen by the demodulator (Y). Distributed applications encounter *sampling* noise: learning the distribution of sensor readings (X) generally involves guesswork if only a subset of all possible data (Y) has been collected.

This paper targets *local* [5] and *semantic* [37] definitions of privacy. Such privacy is provided by a *privacy mechanism*, which essentially constrains the relative strength of noise and sensitive signals in a single device's data. With the *Randomised Response (RR)* mechanism [34], for instance, users (sensors) do not respond directly to a query for some privacy-sensitive property (e.g. (not) having a disease), but instead flip a coin. If tails, they report their true status. If heads, they flip another coin, and report that they have the property if tails, and that they do not have it if heads. This way, individuals retain plausible deniability regarding their true status, but the global proportion of users exhibiting the property of interest can eventually be recovered through statistical means [64].

Other local semantic privacy notions and mechanisms generalise RR to arbitrary inferences and quantified notions of privacy loss. The amount of noise corresponds to the strength of the privacy guarantee. For example, a measurement is said to be ϵ-Locally Differentially Private (LDP) if $Pr(Y|x_i) \leq e^{\epsilon} Pr(Y|x_j)$, $\forall ij$ [62]. Choosing a value for $\epsilon > 0$ is a matter of policy: smaller values imply stronger guarantees, but further obscure aggregate trends [33]. There is no consensus on what signals to consider sensitive, which aggregates to preserve, how to develop mechanisms that balance the two, where that balance (ϵ) should lie, and neither can universally good mechanisms exist [7, 36], giving rise to a zoo of semantic privacy definitions and mechanisms, of which differential privacy (mechanisms) are probably the most well known [6, 33, 53, 59, 62].

2.2 Symbol-synchronous aggregation

Researchers have proposed aggregating data "on the air" through interactions between closely synchronised concurrent transmissions [66]. On additive channels, for example, additive noise can cancel out on the air [4, 20, 25]. The difficulty in establishing such channels limits this approach's practicality [66]: to the best of our knowledge, the fastest demonstration in wireless embedded networking research reports a per-transmitter throughput of 1.6 kbps [26] when computing sums in star topologies, prompting research towards on-the-air computation on less-than-ideal channels [66].

Wired bus networks provide a form of physical-layer computation that is simpler than the computation of sums, namely that of maxima over integers carried in concurrently transmitted frames [8]. Recent work on so-called *Symbol-Synchronous Buses (SSBs)* [47] and similar network architectures [49] demonstrates that this behaviour can be extended to wireless mesh networks (WMNs). SSBs introduce modulation schemes that focus on fast data forwarding rather than reliable data recovery, propagating information through a mesh "like a wave on a pond" [47], effectively realising a fast link-layer broadcast channel on a mesh topology. They thus deliver performance profiles resembling those of wired fieldbuses (e.g. CAN [17]) rather than of WMNs [47]. Zero-Wire, a short-range, 25-node, 4-hop SSB prototype that uses optical and software-defined modulation demonstrates throughput of 19 kbps, sub-millisecond frame delivery across multiple hops with 99% reliability, and computation of on-the-air maxima in 10s of µs per bit involved [46, 66]. End-to-end latency across multiple hops is also close to deterministic; the number of hops that separate sender and receiver affects latency by no more than 10s of µs [66]. The broad impact of fast maxima remains unclear, however: the set of available mathematical machinery that exploits on-the-air maxima for generic computations is smaller than for sums (cf. [3, 46, 66]). We refer to the Zero-Wire paper [47] for a full description of SSBs. In this paper, a set of wireless nodes connected using a SSB can be understood to behave as if they were all physically connected to a single cable pair, enabling fast (100s of µs, i.e. less than the duration of a single frame) and accurate (10s of µs, i.e. a fraction of the symbol duration) synchronisation. [47].

3 APPROACH

This section illustrates Twofer's latency-aware data collection process using a simple example. The example, visualised in Figure 2, assumes each sensor node intends to communicate a single numeric measurement and that some application wishes to compute a histogram over those numbers.

- Before sensor nodes acquire measurements, they organise themselves in disjoint groups referred to as *cohorts*.
- Each sensor node acquires a noisy measurement to be transmitted to a *gateway* device, i.e. a device acting as a central collection point.
- Each sensor lossily quantises the data it intends to communicate to a small integer. The quantisation strategy varies across cohorts, is known to the gateway, and deployed ahead of time on nodes.
- Sets of sensor nodes concurrently transmit the numbers they generated according to a transmission schedule they compute autonomously. The physical layer "'computes" the maximum over concurrently transmitted numbers on the air. These maxima are observed by the gateway.
- The gateway exploits its knowledge of nodes' quantisation strategies to estimate the data that would have been transmitted using a conventional, reliable network. The estimate is fed to upper layers in the network stack as a series of "received" frames.
- The estimated data frames reach the upper layers of the network stack and are therein used for application-level computations. Compared to computations from data gathered conventionally, i.e. using reliable node-by-node transmissions, this lossy data collection approach results in more accurate application-level computations when the sensor data are noisy and latency constraints are strict.

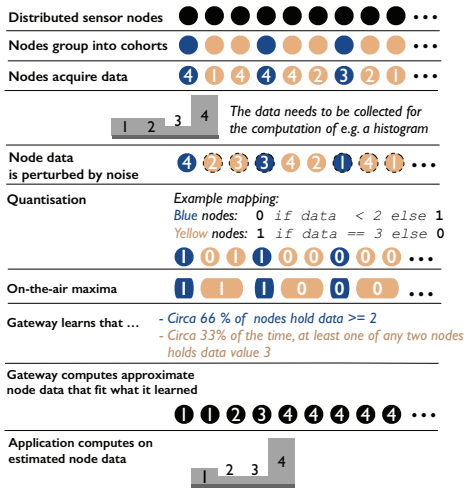

Figure 2: Twofer estimates, rather than losslessly collects, distributed data.

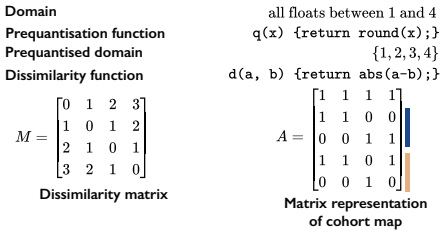

Figure 3: Example values for the concepts used throughout Section 4. The values correspond to the example in Figure 2.

Twofer's on-the-air quantisation strategy optimises for a latency budget and notion of application performance specified by developers. Twofer hence establishes its internal parameters from that developer input, deciding which nodes should transmit at what points in time and how nodes should map their data to small integers. Likewise, the system orchestrates the transmission schedule, and it estimates sensor data from the evidence provided by concurrent transmissions. The concrete methods for doing are detailed in the following sections.

4 OPERATION

Twofer's operation consists of several steps. During a configuration phase, developers define the latency-performance trade-off they wish to optimise, from which Twofer derives its internal parameters, which are then installed on the nodes. At run-time, Twofer orchestrates concurrent transmissions, which take place through a Zero-Wire physical layer. A gateway then estimates sensor data from the transmissions it observes, and existing application code processes the result. The following sections discuss the details.

4.1 Developer input

As with any lossy compression scheme, Twofer's compression strategy depends on the domain, the structure of the underlying data, and the degree of compression required. To define the data domain on which Twofer operates, developers provide a function that quantifies the dissimilarity between any two pieces of data a node might transmit. That is, they provide a *dissimilarity function* `d(a, b)`, where `d == 0` when `a == b`, and `d > 0` in all other cases. We argue that reasonable choices for `d` are always available: obvious choices exist (e.g the absolute value of the difference between two numbers is a measure of how dissimilar they are), as do generic alternatives (e.g. an identity function where `d == 0` if `a == b` and `d == 1` otherwise). Figure 3 shows an example function, as well as examples for other concepts introduced in Sections 4.1 and 4.2.

Twofer requires an explicit enumeration of the data domain, i.e. of the values `a` that nodes might offer to Twofer as input. If the size of the data domain, written "|dom d|", makes such an enumeration impractical due to memory or computational constraints, developers supply a *prequantisation function*, i.e. a function `q` that maps a data value held by a node to another value in a subset of the domain, where the two values are close in the sense defined by `d`. We argue that prequantisation is not a heavy burden for developers: many privacy mechanisms already use small output domains (cf. Section 7.2), subsampling an existing data domain is a generic way to obtain small subdomains, and more advanced techniques exist that represent noisy data using a small output domain [9, 57]. Section 6.1 provides example prequantisation strategies.

Twofer offers developers a list of key-value pairs, informing them of the effect of different configuration options. Each key identifies a Twofer configuration, and each value corresponds to a function of a list of data elements $\mathbf{D}$ and a floating point number t. The function computes, for a specific configuration of which the internal mechanics are hidden from the developer, how Twofer would estimate, after t milliseconds of data collection time, the data in a network for which sensor readings in reality correspond to $\mathbf{D}$. It is up to developers to analyse said functions and to name the desired configuration to be installed on nodes. The simplest possible analysis is one that iterates over the different configurations to identify the one that yields the best accuracy for a given latency budget, where "accuracy" corresponds to the difference between some application-level computation on example data and the corresponding estimate by Twofer. Section 6 provides detailed examples of analyses of Twofer configurations.

Developers use the functions provided by Twofer at configuration time to learn how Twofer interacts with different data distributions, and to what extent the system provides probabilistic guarantees on application performance: the provided functions simulate Twofer's operation. As will be shown in Section 6, such simulations accurately model Twofer's behaviour in physical deployments. As will also be shown, the number of candidate configurations that may need to be evaluated by developer code does not exceed a few hundreds, the performance of candidate configurations exhibits a consistent structure, and each additional call to simulation functions returns in tenths of a second.

4.2 Twofer configuration

Twofer's internal configuration hinges on two parameters: the number of *cohorts*, i.e. the number of different strategies c nodes use to map their data to small integers, and the *frame size*, i.e. the number of bits b these integers occupy. The space of possible settings for these parameters produces the spectrum of configuration options described in the previous section. The parameter space is

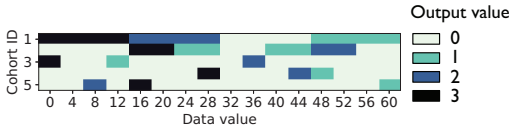

Figure 4: In a cohort map, large output integers are rare. The example shows how Twofer might instantiate the map for an input domain consisting of 6-bit integers.

small: throughout this paper, the number of bits per frame varies from 1 to 4 and the number of cohorts from 1 to 100. Moreover, the parameter space is highly structured: Twofer tunes for smaller latency, for example, by reducing the number of bits per frame.

For each point in the parameter space, Twofer computes a *cohort map*. Such cohort maps detail how nodes map sensor data to integers. Cohorts maps are found using a genetic algorithm, detailed in Section 5.1, and provided to developers as a file to be deployed on nodes and gateways before Twofer is used to retrieve sensor data. A cohort map is a look-up table that, for each cohort, details the integer to which each element from Twofer's (prequantised) input domain should be mapped. The quality of the mapping, optimised for using the genetic algorithm, is quantified as follows.

(1) The system constructs a matrix M, where the element at position i, j is the (developer-provided) dissimilarity between the i-th and the j-th element of Twofer's input domain. M is re-scaled such that the smallest non-zero element equals 1.
(2) The system constructs a second matrix, referred to as A, with as many columns as there are elements in Twofer's input domain. The first row consists entirely of ones. The following rows consist of c sets of 2^b rows, where c is the number of cohorts and b the number of bits per frame. In every set, the k-th row has ones in the columns corresponding to input values that are mapped to $k-1$ for the corresponding cohort, and zeroes elsewhere.
(3) Quality then corresponds to (smaller is better):

$$\left|A^T (A^T)^\dagger \odot M\right|_S + \lambda \cdot \left|\left(\left[0,\ 0,\ 1,\ 2,\ 3, \ldots, 2^b \cdot c - 1\right] \mod 2^b\right) \cdot A\right|_S,$$

where "†" denotes the Moore-Penrose inverse [40], "$\odot$" element-wise multiplication, "$|\cdot|_S$" the sum of all entries in a matrix, and "λ" equals 0.1 (i.e. a scalar considerably smaller than the smallest non-zero dissimilarity value). The first term is proportional to the average dissimilarity between data items that cannot be distinguished in any cohort because of quantisation. The second term regularises: it ensures "large" output integers are used sparingly. The intuition here is that on-the-air maxima let Twofer scan for rare large values across nodes in parallel. As show in Figure 2, gateways then infer aggregate properties describing the distribution of node data from the occurrence of said maxima. These aggregate properties prove more resilient to application-level noise than node-by-node transmissions, explaining Twofer's performance advantages. Figure 4 visualises the distribution of "large" numbers in a cohort map, again using integers as example input data.

4.3 Orchestration

At any given instant, each sensor node belongs to exactly one *cohort*, and nodes use the mapping strategy of the cohort to which they belong to map their measurements to integers. Each cohort is associated with an integer that details the number of concurrent transmitters for that cohort. Setting this number too low or too high would cause gateways to almost always see a zero or non-zero transmission respectively and hence slow down learning, since frames interact on the air to compute a maximum over the transmitted frames, and since cohort map design ensures that frames only rarely carry "large" numbers. The number of concurrently transmitting nodes for the c-th cohort, μ_c, is hence set according to the following heuristic, which aims to ensure that gateways see a zero transmission roughly half of the time by increasing the number of transmitters along with the number of zero-mapped data values:

$$\mu_c = \underset{m \in \mathbb{N}, m \geq 1}{\operatorname{argmin}} \left| 0.5 - \left(\frac{\sum_{j=1}^{|\mathrm{dom\ d}|} A_{2^b(c-1)+2,j}}{|\mathrm{dom\ d}|} \right)^m \right|.$$

Transmissions are organised in *superframes*. During a superframe, a Twofer gateway requests non-overlapping subsets of sensor nodes to transmit. For this purpose, sensor nodes are assigned unique and sequentially increasing identifiers. At the start of a superframe, the gateway informs nodes of network metadata, namely a random seed and the identity of nodes joining or leaving the network. Nodes compute the cohort they belong to and the moment they should transmit as follows. At the start of a superframe, they randomly shuffle the list of sequential node identifiers in the same way by using the shared random seed that was established by the gateway. Nodes then partition the list: they cycle through cohorts, each time popping a number of nodes from the list that is equal to the number of concurrent transmitters for the selected cohort, until the list is exhausted. A group of nodes popped from the list together is referred to as a *turn*: nodes in the same turn transmit concurrently, and transmissions progress turn-by-turn. Gateway requests instruct nodes on a range of turns to transmit. As network latency on an SSB is highly deterministic [47], nodes that belong to one of the requested turns compute when to transmit from the number of bits yet to be transmitted before them. The exact synchronisation mechanism, which ensures that nodes in a turn tranmsit concurrently, is detailed in Section 5.2 and heavily exploits SSBs' deterministic time behaviour, as documented in [47]. Throughout this paper, turns comprising 1 to 10 nodes are typical.

4.4 Estimating distributed data

Gateways maintain a set of histograms in which they record the fraction of times each integer occurs as the maximum of concurrent transmissions for each cohort. Transmissions do not carry addressing information; gateways rely on the time offset between their request and observed replies to establish the cohort from which replies originate. When all sensor nodes have transmitted, the gateway computes an estimate for the data held by nodes from its collection of histograms. This estimate is a set of sensor values that approximates (according to d) the set of data that would have been transmitted had nodes directly conveyed the data they held, without first encoding it through a cohort map.

Gateways estimate the network-wide data distribution by solving a linear system of equations. Its unknowns correspond to the fraction of nodes holding a given element from Twofer's input domain, and hence by definition sum to 1. The right-hand side of the system estimates, for every cohort and for every value in $\left[0, 2^b - 1\right]$, the probability that a node in that cohort transmits that value by correcting the gateway's observations for the effect of the

on-the-air computation of maxima. The system reads as follows, writing f_{cv} for the fraction of maxima equal to v for the c-th cohort, and x_j for the (unknown) fraction of nodes holding the j-th data value in the input domain:

$$\sum_{j=1}^{|\text{dom d}|} A_{2^b(c-1)+k+2,j} \cdot x_j = \left(\sum_{v=0}^{k} f_{cv}\right)^{1/\mu_c} - \left(\sum_{v=0}^{k-1} f_{cv}\right)^{1/\mu_c},$$

$$\forall \text{ cohorts } c, \forall \text{ integer frame values } k \in \left[0, 2^b - 1\right].$$

Each equation relates the frequency with which values mapping to k occur in a particular cohort to the network-wide data distribution. The number of such equations is hence determined by the number of cohorts and the number of bits per frame, and it is in general far smaller than the cardinality of Twofer's input domain. The gateway's estimate for the data held by all nodes therefore is the result of extrapolating from the learnt parameters: gateways assume a uniform distribution over domain elements that cannot be distinguished. Twofer does not consider configurations for which the number of linearly independent equations exceeds |dom d|: the number of parameters recorded by the gateway then exceeds the number being estimated.

The system of equations solved by the gateway is underdetermined, its unknowns are non-negative, and its equations are linearly dependent. Twofer hence regularises the system by adding the following equations (i.e. ridge regression [35]):

$$x_j = \kappa, \forall j \text{ in the enumeration of dom d},$$

where κ is an arbitrarily chosen "small" constant (we set $\kappa = 10^{-5}$). Twofer then solves the combined system as a non-negative least squares problem (cf. Section 5.1). The solution, i.e. an estimate for the fraction of nodes holding each element of the input domain, is then converted to a list of data that can be fed to gateway application code: Twofer multiplies the estimated fractions by the number of nodes in the network, rounds, and it constructs a list of "received" frames by adding each data element as many times as the rounded count[1]. Application code is unaware that the resulting list of data was obtained using Twofer: reconstructed frames are propagated to the application layer as if they were conventionally received[2].

Erroneous transmissions propagate to the application layer: some or all nodes may miss the gateway's request, some or all replies may be corrupted, and synchronisation may go wrong. Gateways associate a received frame with a specific cohort by comparing the time at which it is received with the expected times, and selecting the closest match. The effect of these design choices is examined in Section 6.2.

5 IMPLEMENTATION

5.1 Computational aspects

Twofer solves two inference problems. The first is the least-squares problem detailed in Section 4.4. Efficient algorithms for this problem exist; we use SciPy's implementation [2] of the Lawson-Hanson algorithm [39]. The second problem is that of finding good cohort maps, discussed in Section 4.2. Twofer tackles this problem by greedily and iteratively using a genetic algorithm, as detailed below.

(1) Twofer first initialises an empty set, which will hold the part of the cohort map for the cohorts considered so far.
(2) For each cohort c:
   (a) Twofer initialises a population of random mappings between elements of Twofer's input domain and integers in $\left[0, 2^b\right)$, where b is the number of bits per frame.
   (b) Twofer adds each member of the population to a copy of the set, and evaluates the resulting partial cohort map using the quality function in Section 4.2.
   (c) Twofer improves the population through a simple genetic algorithm provided by PyGAD [19][3]. Optimisation continues until convergence or until a maximum number of generations is reached (cf. Section 6.3). Next, the best mapping is added to the set of mappings established so far.

Section 6.3 evaluates the computational overhead associated with these two inferences. Sensor node overhead is extremely limited: cohort maps are small (no more than few kB's throughout this paper), and computational overhead is dominated by the need to compute the transmission schedule (< 10 microseconds per turn).

5.2 Network aspects

Twofer runs on top of Zero-Wire [47], a symbol-synchronous bus implemented on software-defined transceivers hosted on microcontrollers (namely STM32G474s [55] and AM335x PRUs [56]). We extend the current C implementation to enable the synchronisation required for a series of concurrent transmissions: Zero-Wire research so far focuses on scenarios in which a single request follows a single reply [46, 47]. The transceiver interface offered to applications is extended with a function `deferred_reply(input, output, delay)`, through which applications program the transceiver to automatically send a frame carrying `output` at the first point in time where the channel becomes idle after receiving a frame carrying `input` and waiting for the specified `delay`. Upon receiving a request, Twofer nodes transmitting in a requested turn set this delay to 100 µs less than the time it takes for all preceding turns to transmit back-to-back through the bus. The bus channel becoming idle at the end of a transmission associated with one turn thus synchronises the transmissions of nodes in the next.

6 EVALUATION

Twofer is evaluated in four parts. Section 6.1 simulates the system's effect on a selection of LDP mechanisms, investigating whether Twofer can be applied generically. Such mechanisms communicate large, easily quantified amounts of noise, from which Twofer's performance drivers can be understood. Section 6.2 hones in on one example mechanism to provide a detailed account of Twofer's performance on a Zero-Wire test bed. Experiments examine performance in the presence of transmission and synchronisation errors, whether expectations from simulation hold up, if and how the system can tune data collection for latency-bound performance, and

[1] This approach faces the same challenges as when dividing parliament seats proportionally with votes (cf. [21]). For privacy mechanisms, which generally do not consider the number of nodes, Twofer pretends that number is 10^2 times larger than it is.
[2] "Data" for Twofer correspond to transmission payloads. Since privacy implies that nodes are interchangeable, link-layer metadata are not reconstructed.

[3] Configuration: population of 200 members; cohort maps are represented as lists of integers denoting Twofer's output value for each input value; 4 parents are selected and mate with single-point crossovers; mutations include swapping the inputs associated with a set of outputs and randomly changing output for a given input.

	Mechanism characteristics				Twofer settings (1 bit per frame)			Evaluation setting ($\epsilon = 1$)
	Domain (baseline encoding)	Estimation problem	Loss func.	Details	Cohorts ("more param." / others)	Prequantisation	Dissimilarity func.	Hyperdistribution for right half of Figure 5
RR	Categorical (binary encoding)	Histogram (normalised)	L^1 distance	6 domain elements; baseline counted as $\lfloor \log_2 6 \rfloor = 2$ bits (not 3) per frame to prevent pro-Twofer bias	4 / 2	None	Identity (cf. Section 4.1)	Uniform on L^1-sphere [10] , i.e. no assumptions on data distribution
Laplace	**Integers (16-bit, signed)**	**Median**	**Abs. error**	**Values clipped to range; intermediate floats rounded**	32 / 16	**Map to nearest of 64 evenly spaced ints across range**	**Abs. error**	**Randomly weighted mixture of 3 Gaussians** $\mathcal{N}_i$**,** $\mathcal{N}_i\left(\mathrm{Unif}\left(-2^{15}, 2^{15}\right), \left(2^{16 \cdot \mathrm{Unif}(0,1)}/3\right)^2\right)$
OLH	Categorical (binary encoding)	Histogram (normalised)	L^1 distance	32 domain elements; one fixed hash function	3 / 2	None	Identity	Uniform on L^1-sphere
PrivSet	Sets (binary mapping between possible sets and numbers)	Fraction of nodes observing some event	L^1 distance	$d = 32, m = 4, k = 1$ (defined in [59])	32 / 16	Built-in ($k = 1$, [59])	If comparing two dummy or two non-dummy [59] elements: identity, else: 1000× identity	Uniform on L^1-sphere

Table 1: Parameter setting for Section 6.1. The settings marked in bold are examined in further detail in Section 6.2.

whether such tuning is robust. Section 6.3, details the system's computational overhead. Section 6.4 studies its application in a smart grid context. Finally, Section 6.5 discusses the presented results.

6.1 Generality

Scenario. Twofer is evaluated on a selection of four LDP mechanisms: Randomised Response (RR) [34], Laplace [14], Optimised Local Hashing (OLH) [60], and PrivSet [59]. This selection ensures a degree of diversity in terms of input data types (numbers, sets, categorical data), size of the data domain (few elements, tens of thousands of elements), inference targeted (scalar statistics, histograms, other multivariate statistics), and mechanism complexity (from small and seminal [14, 34] to larger and more subtle [59, 60]). To avoid the introduction of unnecessary complexity, no multi-round LDP protocols (e.g. [44]) were selected: nodes transmit once and do not change their behaviour based on another node's data. Parameter setting for the selected mechanisms has been extensively studied before [33, 59, 60], so the experiments in this paper suffice with analysing Twofer's performance in "reasonable" (i.e. non-extreme and reportedly "good") settings, given in Table 1.

The experiments detailed in this section quantify Twofer's impact on latency-accuracy trade-offs encountered when collecting data pre-processed by the selected mechanisms. Such trade-offs are aggregate properties of multiple experimental *runs*. For each run, an underlying data distribution is first drawn from a given *hyper-distribution* (i.e. (non-)parametric distribution of distributions, cf. method in [6]). Nodes then draw the data they are about to communicate from that distribution and apply a privacy mechanism to the data. The same "private" data is then collected using both Twofer and a baseline approach that uses conventional transmissions carrying conventionally represented private data (e.g. the shortest possible binary encoding for categorical values, binary integers for numbers). At every stage of this process, the data collected so far, which is either a set of noisy node data for the baseline or an estimated version thereof for Twofer, is fed to the privacy mechanism's recovery algorithm. As explained in Section 2.1, the result is an estimate of a parameter, the recovery of which the mechanism under question was designed for. The accuracy of this estimate corresponds to the *loss* between the estimate and its actual value computed on the distribution from which the data was drawn, for some loss function. Table 1 summarises the hyperdistributions, loss functions, and data representations used in this section.

Each run results in a series of points relating the amount of data collected so far to the loss for a parameter estimate based on that data. Across runs, the p-th percentiles of loss values yield a curve showing a latency-accuracy trade-off (cf. [6]). The horizontal distance between two such curves corresponds to the traffic reduction to reach a given accuracy level with a certain degree of confidence by means of a data collection method (cf. Figure 1). Percentiles of the vertical distance between curves associated with a single run are a measure of the accuracy improvement attained under a traffic bound. The role of the parameter that configures the privacy level for which those curves are obtained, i.e. ϵ, will be studied in detail in the following section and is for now set to a "reasonable" value of 1 (cf. [6, 16, 59, 60]). The Twofer configurations underlying this analysis can also be found in Table 1, for now without further detail: the following section shows that good configurations can be found automatically, that they can be robustly exploited on real hardware, and it examines the relationship between e.g. the available latency budget and Twofer's internal paraneters.

Results. Each mechanism under analysis corresponds to a row of graphs in Figure 5. The first graph from the left in each row shows latency-accuracy curves for Twofer, for the baseline, and for two Twofer variants discussed later, with Thompson confidence intervals [28], and where data is drawn from a point distribution (i.e. where noise in the underlying distribution is minimised). Twofer improves the trade-off between latency and median accuracy: the curve for Twofer in the leftmost column of Figure 5 is consistently below that for the baseline when the amount of collected data is sufficiently small. This improvement is a form of distributed compression: compared to the baseline, Twofer collects less data (its curve extends less far to the right), does not reach the same level of accuracy eventually (extends less far to the bottom), and performance improvements disappear when plotted in function of the number of nodes from which data is collected rather than the collected amount of data (cf. second column of the figure).

Twofer can be applied in a variety of settings, ranging from the collection of a few bits of information from a few nodes (as shown for RR and OLH), to that of several thousands of bits from equally many nodes (PrivSet). Yet, the size of its performance advantages varies considerably across configurations, mechanisms, latency budgets, and accuracy targets. The left half of Figure 5 illustrates this point by showing latency-accuracy curves for an inappropriately configured number of cohorts (the "Twofer (more param.)" variant, details in Table 1) and for a configuration in which concurrent transmissions are disabled ("no concurrent"). Concurrent transmissions and parameter tuning prove essential: without them, performance advantages may disappear altogether.

Twofer configurations optimise for a specific latency-accuracy target: comparisons of Twofer and "Twofer (more param.)" show

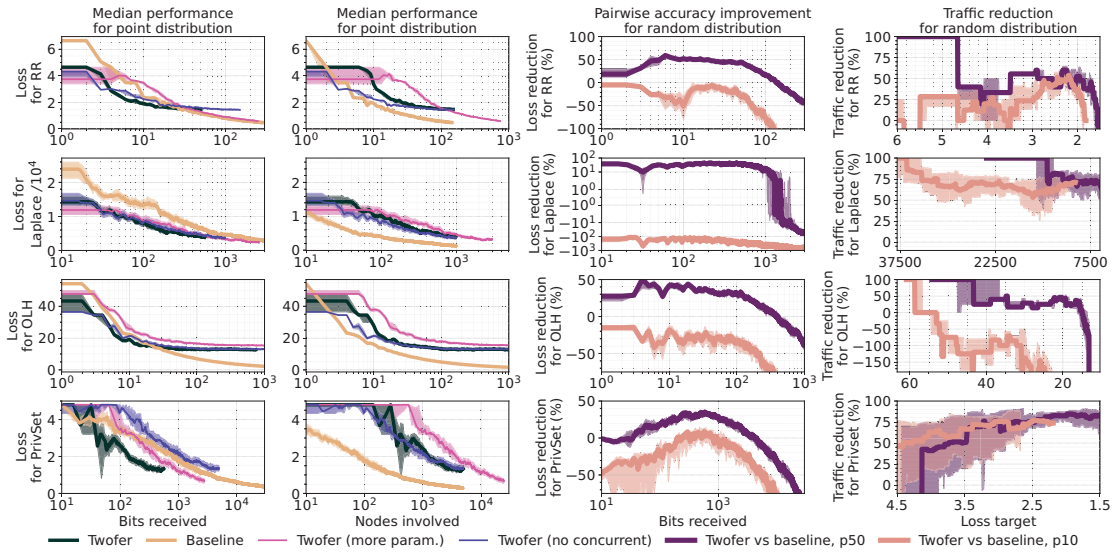

Figure 5: Concurrent transmission and latency-bound optimisation, applied generically to semantic privacy mechanisms, typically result in a circa 30% accuracy gain / a 25-75% traffic reduction.

that configurations that prove suboptimal when little data can be collected or fewer nodes are available, may prove advantageous when the converse is true. The first and second column of plots in Figure 5 purposefully evaluate "Twofer (more param.)" on a network with more nodes than used for the other data collection approaches to illustrate this phenomenon. We point out that Twofer's performance advantages apparently increase with the difficulty of the underlying estimation problem: for Laplace and PrivSet baselines, relative accuracy loss decreases much more slowly with the amount of data collected than for the other two mechanisms, yet Twofer's performance advantages appear larger and extend over a larger range of accuracy-latency settings. This observation implies that Twofer's performance advantages extend beyond its application to privacy mechanisms: all mechanisms under analysis use the same privacy level ϵ, but the size of Twofer's performance advantages varies considerably. The cause is Twofer's mitigation of relatively large amounts of sampling noise for Laplace and PrivSet due to these mechanisms' relatively large domains.

The right half of Figure 5 provides a more detailed and generic point of view on the effects described above, by experimenting with larger classes of underlying data distributions (details in Table 1), and by detailing the distribution of performance effects. The third column in the figure shows that Twofer's performance advantages are an aggregate property: pairwise, relative comparisons of loss measurements associated with single data collection run sometimes favour the baseline approach (e.g. for the 10^{th} percentile of accuracy comparisons). For example, "lucky", near-zero random noise for the Laplace mechanism can lead to a very accurate baseline estimate, which fundamentally cannot be achieved by Twofer because of its use of quantisation. More often than not, however, Twofer wins: median values for such a comparison show an accuracy improvement in the 10s of percents. As discussed above, Twofer must be tuned for a specific setting: if there is sufficient time to collect more data than optimised for in a given configuration, performance advantages may disappear. When considering the amount of data collection required to reach a given accuracy level (fourth column), Twofer demonstrates a 25-75% traffic reduction, even if such accuracy guarantees must be given with high confidence. One caveat applies: Twofer's advantages on mechanisms that output a hash (e.g. OLH) decrease rapidly with the amount of data collected. The reason is Twofer's reliance on developer hints about the structure of node data, whereas hash functions by definition destroy such structure in a way that cannot be undone without breaking black box semantics, i.e. inverting the hash.

6.2 Detailed performance

Scenario. The experiment marked in bold in Table 1 is repeated on the 25-node, 4-hop Zero-Wire test bed detailed in [47]. The Laplace mechanism is reported on because of its popularity and seminal character [14]; experiments on other mechanisms display similar trends as those discussed below. As in [46], the Zero-Wire network is configured to run at a symbol rate of 19 kilobaud. To analyse Twofer's effects in networks large enough to make accurate inferences under stringent privacy constraints, transmissions from 400 virtual nodes are scheduled on top of the 25 physical nodes, while ensuring that co-located virtual nodes are never scheduled to transmit following the same gateway request (cf. Section 4.3).

The results below report accuracy-latency curves obtained for a variety of configurations that a developer might select: ϵ is set to 0.1, 0.3, 0.5, 1, 2, 3 (cf. [6, 16, 59, 60]); the number of cohorts to 1, 2, 3, 5, 8, 10, and the number of bits per frame to 1, 2 or 3. To reduce experimentation time, each curve is obtained by first calculating the empirical mean of loss values for a latency budget across 20 runs, and then plotting the smallest mean loss value that is not exceeded for latency values larger than the chosen budget (thus ensuring monotonous curves in the presence of statistical noise). As in Figure 1, latency savings then correspond to the horizontal distance between these curves, and accuracy improvements to the vertical distance between them. For the purpose of this section, "latency" refers to the time spent by the gateway to receive data frames, including

all per-frame overhead (e.g. preambles, inter-frame delays), but excluding the time spent receiving gateway requests.

Results. If developers are to use Twofer successfully, they should be able to select a "good" configuration using the simulation functions described in Section 4.1 and then to capitalise on that knowledge in an actual network. Figure 6 shows that that is indeed possible: in the explored configurations, simulation and test bed results are consistently close, in the sense that any performance discrepancy between the two (due to losses or data corruption in the "real" network) is consistently far smaller than the performance advantages offered by Twofer relative to the baseline. The figure illustrates this fact using a selection of latency-accuracy curves for Twofer executed on Zero-Wire, Twofer executed in simulation, and simulation of the baseline. Each plot in the figure shows curves for a specific parameter setting for Twofer, detailed above it; curves drawn in the same style within a single plot correspond to different values for ϵ (from top to bottom: 0.5, 1, 3). The figure deliberately shows a simulated baseline (i.e. one that assumes no transmission errors), since the resulting comparisons are maximally disadvantageous to Twofer: if Zero-Wire's unconventional characteristics would somehow artificially disadvantage conventional approaches, such characteristics cannot lead to an overstatement of Twofer's effects. Simulated and actual latency is consistent down to 10s of µs because of Zero-Wire's low-level structure [47].

Figure 7 characterises the effects of Twofer's internal parameters on the system's performance, revealing that the configuration options to be considered by developers are small in number in high-privacy regimes and highly structured. The top row interprets "optimal parameter setting" as the number of bits per frame and number of cohorts that produces the most accurate results when the amount of data that can be collected is limited and ϵ is set to some fixed value. Such data collection budgets are shown in units of bits for visual clarity: the range in the figure (2^3 for 1-bit frames to 2^9 bits for 3-bit frames) corresponds to 3.8–117 ms of latency. The bottom row takes a dual perspective, reporting on the configuration that ensures a level of accuracy is met as fast as possible, i.e. using as little data as possible. The leftmost plot for every row in the figure details the optimal number of bits per frame. This number decreases as the available latency budget decreases, as the amount of noise increases, or as increasingly coarse-grained estimates suffice, with frames carrying only a few bits of data proving optimal under reportedly sensible privacy regimes. Part of the bottom-left plot is left blank: it is impossible to learn anything to arbitrary accuracy from a finite amount of noisy data, irrespective of the system that is is used to do so, and noise limits achievable accuracy.

The number of cohorts, studied in the middle column of Figure 7, provides a more fine-grained way to tune for fast, coarse-grained approximation: within the region associated with a fixed number of bits in the left column, the optimal number of cohorts in the middle column increases when moving to the bottom right of the plot. To better visually separate such regions, the one associated with 2-bit frames in the left column is left blank for the plots in the middle column. The number of cohorts also lets developers tune for robustness: the right column in Figure 7 displays how the grids in the left two columns evolve when optimising for a subset of experimental runs, rather than all 20. Averaged across the grid, such local optimisation increases the number of cohorts, but does

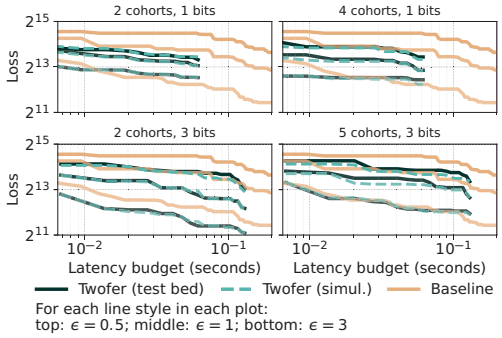

Figure 6: Simulated performance improvements carry over to a Zero-Wire test bed, enabling automated parameter tuning.

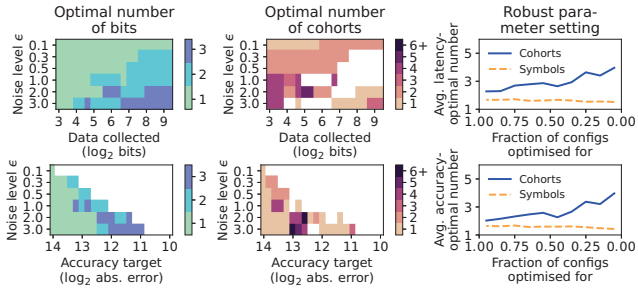

Figure 7: As a rule of thumb, Twofer uses 1-bit frames and 1–5 cohorts in high-privacy regimes.

not considerably affect the optimal number of bits per frame. When developers select Twofer configurations that are more robust to changes in the underlying data distribution, Twofer thus internally relies on an increased number of cohorts.

Figure 8 examines the relationship between scenarios Twofer might optimise for and its performance advantages in the corresponding configuration. Relative accuracy improvements (i.e. loss reductions; left plot) appear relatively consistent across latency budgets and privacy regimes: in line with simulation results, accuracy improves by tens of percents; only at the edges of the explored parameter space does our analysis not find a Twofer configuration that improves upon the baseline. Latency reductions (right plot), however, increase as privacy levels increase, they are larger when estimates can be more coarse-grained, and they exceed a 16× improvement in the strictest privacy regimes. For any point in the explored parameter space, there exists some Twofer configuration that is optimal, except for the points involving the strictest accuracy requirements and laxest privacy guarantees: Twofer configurations form a continuum, in which the optimal number of bits per frame gradually increases towards that of baseline approaches as noise levels decrease, and where such parameter tuning partially mitigates the adverse impact of noise on application-level latency. Such parameter tuning cannot overcome the accuracy impact of noise in the same way: for any conceivable system, noise limits the accuracy to which latent parameters can be learnt from a given amount of data, which is again visualised as a blanked-out region in the right half of the right plot in Figure 8. A region on the left side of that plot, corresponding to what can be learnt in less than 5 ms, is also left blank, for visual clarity: latency measurements for these performance settings are coarsely discretised since they

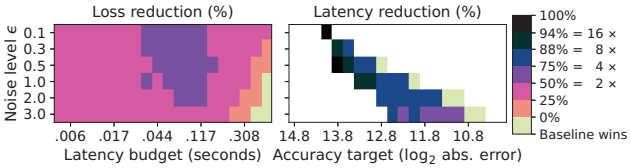

Figure 8: Latency reductions increase with the privacy level. $2-4\times$ latency reductions are typical.

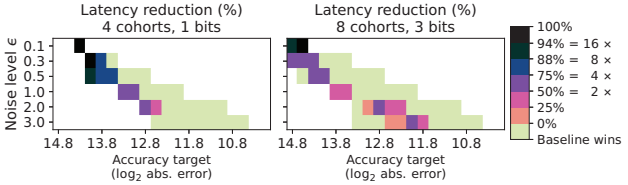

Figure 9: Twofer's performance advantages increase with the privacy level, even if wrongly configured.

can be achieved by collecting only a few frames, causing latency reduction measurements to not show clear trends.

The extent to which developers can correctly identify "good" Twofer configurations may remain limited due to an imperfect understanding of the underlying data. Figure 9 therefore illustrates Twofer's performance when the system is wrongly configured: the number of cohorts and bits per frames are set to a fixed, arbitrarily selected, and generally suboptimal value. The two plots show that such a configuration decreases the size of Twofer's performance advantages relative to those in Figure 8, and that it reduces the size of the privacy-accuracy space in which applying Twofer proves beneficial. Still, the introduction of privacy guarantees motivates the design of Twofer. Concretely, the system's performance advantages increase with the privacy level, and so does Twofer's robustness against misconfiguration: settings in which Twofer outperforms the baseline occupy an increasingly large part of the achievable privacy-accuracy space as noise levels increase.

6.3 Computational overhead

Scenario. This section quantifies the computational overhead incurred because of the two inference problems introduced by Twofer: finding "good" cohort maps to be installed on nodes (henceforth "training"), and solving the least-squares problem that converts per-cohort histograms into the estimate for data held by nodes (henceforth "gateway inference"). The experiments below detail the time it takes to solve these problems on a machine with 16 Intel Xeon Gold 6140 cores at 2.3 GHz for training, and a Raspberry Pi 4 [1] for gateway inference. The structure of the inference problems, i.e. the dissimilarity function for training, and the cohort map and collected data for gateway inference, prove to have no considerable impact on compute times in our experiments. For gateway inference, measurements correspond to average wall clock times for execution on otherwise idle processor cores. For training, reported times are wall clock convergence times, i.e. time bounds within which at least half of 10 runs of the algorithm in Section 5.1 converge to within 2.5% of the best observed fitness value. If convergence is not achieved in 300 generations per iteration of the algorithm, the figures discussed below report "N/A".

Results. Figure 10 shows the time necessary for gateway inference, relative to the size of the (prequantised) domain, the number of

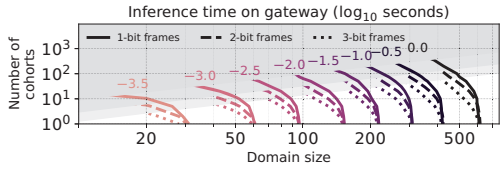

Figure 10: Gateways incur $< 10^{-2}$ seconds of computational overhead for configurations shown relevant in this paper.

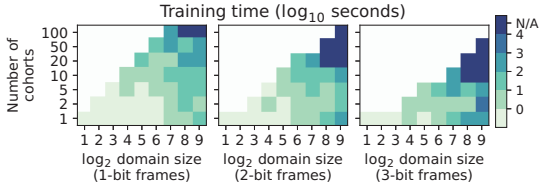

Figure 11: Computing how to use Twofer for a specific application takes circa 10^3 seconds.

cohorts, and the number of bits per frame. Inference time increases along with those parameters, yet all estimates made for experiments detailed earlier in this paper can be computed in less than 10^{-2} seconds. Problems several times larger than those discussed in this paper can be solved in less than a second, and compute times for a given configuration prove at least an order of magnitude smaller than the timescales necessary for data collection (cf. Figure 6). The top left of the figure is not filled in, as the corresponding configurations generally estimate "too many" variables (cf. Section 4.4).

Figure 11 details training time, showing a separate plot for each setting for the number of bits per frame: separate plots improve visual clarity in the presence of statistical noise due to the stochastic nature of the underlying genetic algorithm. Again, compute time increases with the problem size. Prequantisation, detailed in Section 4.1, proves effective in controlling the computational burden: compute times decrease rapidly with the domain size. For all configurations studied earlier in this section, a cohort map can be found in less than 10^3 seconds, and configurations with parameter settings several times larger than those studied here can still be obtained in less than 10^4 seconds.

6.4 Case study

Scenario. This section applies Twofer to latency-sensitive load monitoring in a smart grid context. In such applications, the instantaneous power consumption of electrical appliances is measured continuously and reported upstream to facilitate e.g. demand response [63]. Application-level latency requirements vary from tens of milliseconds to a few minutes [63]. To mitigate privacy concerns, Laplace-distributed noise can be added in the frequency domain of individual devices' signals before transmitting said time series data [48]. This paper studies the SynD power monitoring data set [38], reporting on 22 devices that each sample their power consumption at 5 Hz. The goal is to approximate the sum of the individual AC power traces at a gateway, while minimising the maximum error over time [63]. The following paragraphs discuss how to achieve this using a baseline and a Twofer-based approach.

Baseline. Since electrical appliances' power data is sparse in the frequency domain [42], developers can reduce traffic and latency by transmitting frequency-domain data. The presented baseline hence

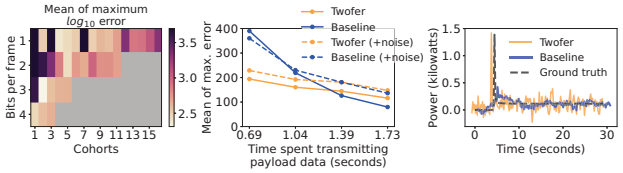

Figure 12: Twofer improves distributed power peak tracking under latency constraints.

computes cosine transforms over randomly selected 30-second windows in the data set. Upon inspection of the data set, a developer might notice the magnitudes of the frequency components are approximately log-uniformly distributed between 0.25 and 5500 W, and hence, depending on the remaining latency budget, quantise frequency-domain components to Q bits in a logarithmically evenly spaced series of 2^Q points in that interval. The analyses in the following paragraph consistently report on the baseline approach (i.e. value for Q) that transmits the most data under a given latency budget, further favouring the baseline by assuming its data is transmitted without frame losses.

Twofer. Using the same domain knowledge as for the baseline, a Twofer developer's configuration workflow looks as follows.

```
// First, the developer specifies domain knowledge, cf. baseline
T = Twofer()
T.enumerated_domain = [sign * 0.25*(5500/0.25)**(i/15)
                       for sign in (-1, 1) for i in 0..15]
T.prequantify = function(x) { //map to closest in enumeration
                  return T.enumerated_domain[argmin(
                           [abs(x-y) for y in T.enumerated_domain])] }
T.dissimilarity = function(x,y) { return abs(x-y) }
// Developer specifies optimisation goal (arbitrary example)
T.bus_speed = 9.5 //kbps
latency_budget = 1.04 //seconds of payload data
// Next, the developer searches the optimal config.
// We assume "training_data" is a set of examples, wherein each
// example is a list of the data values held by nodes.
// Likewise, "App.error(training_data, estimated_data)" computes
// the sum of inverse frequency transforms on a ground-truth and
// estimated data set and returns the mean maximum error between both.
for num_bits, num_cohorts in Twofer.possible_configs(
  max_bits=4, max_cohorts=16): //arbitrary limits
      estimated_node_data = [T.simulate(latency_budget, D,
        num_bits, num_cohorts) for D in training_data]
      error = App.error(training_data, estimated_node_data)
      //The errors are visualised in left graph of Fig. 12
//Based on error analysis, the developer picks the best config
result = T.get_cohort_map(bits=3, cohorts=2)
```

The developer installs the resulting cohort map on the gateway and sensor nodes. On sensor nodes, Twofer organises the network as a pool of frequency-domain sensors, that is, the system is used to estimate the distribution of weights for every frequency bucket independently. The gateway thus recovers a set of magnitudes for every frequency and feeds that data to the application layer, which, as for the baseline, recovers time-domain data.

Comparisons between Twofer and the baseline are skewed against Twofer: none of today's wireless technologies transmit frames of a few bits efficiently. Likewise, this study consider 30-second windows, making the baseline benefit from frequency-domain sparsity. Twofer's application on smaller time-domain windows improves its relative performance, as it does not benefit from frequency-domain sparsity to the same extent as the baseline (i.e. frequency coefficients being near-zero and thus quantising near-exactly for the baseline). Contemporary load monitoring approaches are moving towards such shorter time windows [63].

Results. The middle graph in Figure 12 shows the result of simulating baseline- and Twofer-based communication using the configuration found above, for various latency budgets. To demonstrate robustness, we artificially induce 5% frame loss for Twofer sensor nodes' messages. For latency budgets exceeding circa 1.3 seconds, the baseline approach wins; for those below, Twofer wins. Adding Laplace noise in the frequency domain (here with variance 100 watt^2) shifts the cross-over point to the right, thus revealing the same trends as in the previous sections. The reason for Twofer's better performance in constrained settings is illustrated in the rightmost subfigure: maximum errors are driven by protocols' ability to accurately communicate the amplitude and location of brief power peaks. Twofer's ability to monitor many nodes in parallel makes it easier to accurately detect peaks that emerge at one of many nodes, or as the result of simultaneous behaviour by many nodes, compared to baseline approaches.

6.5 Discussion and limitations

Developer overhead. Twofer imposes a larger amount of configuration effort on developers than conventional network stacks because it optimises for a specific, developer-provided latency bound. Evaluation results, however, confirm that this burden is kept to a minimum. In particular, developers must only define what latency budget the system should optimise for and what constitutes "good" performance, using the interfaces detailed in Section 4.1. Twofer's internal configuration is established automatically from this specification, i.e. through simulation-based analysis rather than trial-and-error on real networks. Doing so is possible because the performance difference between idealised and real-world Twofer proves small relative to the size of the performance advantages that the system introduces, and because the effects of parameter setting on (real-world) Twofer follow predictable patterns (cf. Section 6.2). Future efforts can further reduce the burden imposed on developers by, for example, providing a catalogue of ready-made dissimilarity functions suitable for specific application domains.

Physical-layer portability. Twofer relies on on-the-air aggregation functionality not found in off-the-shelf transceivers. Nevertheless, on-the-air aggregation has been demonstrated multiple times in wireless systems research, across a variety of deployment scenarios [3, 20, 46, 52]. All these efforts remain limited by the lack of mature platforms: further research is necessary to understand how fast and accurate on-the-air aggregation might become, and how such advancements impact system-level performance.

7 RELATED WORK

7.1 Generic applicability

Twofer differs from existing approaches towards on-the-air aggregation by introducing an empirically validated formalism that improves latency-accuracy trade-offs for arbitrary applications characterised by high semantic noise levels at the level of sensor data.

Existing theoretical frameworks (e.g. compressed sensing [11], nomography [23]) detail how on-the-air aggregation on idealised additive channels may speed up the computation of almost all functions of distributed data. While practical implementations of such channels have shown useful in specific scenarios (e.g. for wildfire detection [20]), they can only calculate on-the-air sums to a limited

degree of accuracy [20, 26, 31]. Such limitations currently leave unclear to what extent the generic applicability of those frameworks can be reconciled with real-world channel conditions, particularly when latency constraints prevent the time-consuming error correction schemes proposed by current approaches [20].

Several authors have advocated for the calculation of maxima rather than sums to improve the practicality and ease the analysis of on-the-air aggregation [3, 52]. Still, it has remained unclear to what extent such maxima serve data collection needs for computations beyond those of simple arithmetic aggregates like sums, maxima, and counts [3, 52]. A notable exception is One-Take [46], which exploits maxima to accelerate real-time binary classification. In particular, the One-Take paper relies on Bayesian arguments and "simple" (i.e. single-parameter) decision problems to demonstrate the *existence* of concurrent transmission protocols that reduce latency. Twofer, by contrast, shows how protocols suitable for maximum-based estimation can be *constructed* and used to inform a more complex (i.e. multi-parameter) class of inference problems.

Some on-the-air aggregation mechanisms, e.g. QuAiL [20], reportedly *provide* privacy, essentially because concurrent transmissions obscure an individual node's data [29, 50]. Other studies show that such an approach is only secure relative to arguably weak attacker models [30, 54]. Twofer is hence not designed to itself provide privacy but studies the exploitation of existing guarantees at the network level, while preserving them (cf. [45]).

7.2 Optimised communication

Twofer is not the first system to argue for much smaller frames and hence much more coarsely quantised transmissions than commonly found in contemporary network stacks to optimise communication communication efficiency, but it is the first to examine the effects of latency constraints on such link-layer optimisations and to provide robust tools for latency-aware (i.e. non-asymptotic) optimisation.

Several LDP mechanisms adjust their output domain size with respect to a privacy level in order to reduce asymptotic communication costs [6, 60]. Theoretical analyses also confirm that small frames suffice to yield close to optimal asymptotic network performance in high-privacy regimes [9, 60], and that optimal frame sizes decrease as the strength of privacy guarantees increases [13, 57]. Moreover, generic constructions that convert privacy mechanisms into ones communicating only a few bits per node have been described, e.g. by incorporating a shared random seed [9, 57]. The basic concept underlying such constructions, diversified ("dithered") quantisation, has been studied for decades [65], and continuously re-appears in the sensor privacy literature in an applied form, e.g. for federated learning with ad-hoc privacy guarantees through *masking* [41], or for privacy-preserving detection of popular websites [16]. Considering small-frame protocols outside of privacy research, e.g. Dabeer et al. [15] provide an information-theoretical account of maximum-likelihood estimation for distributed signals through 1-bit, quantised, dithered transmissions from sensor nodes. All of the above approaches optimise the internal operation of privacy or data-producing mechanisms. Twofer, by contrast, examines the performance gains attained by optimising low-level networking, treating such mechanisms as a black box.

Twofer's account of latency budgets also distinguishes it from some of the previously discussed on-the-air aggregation systems: e.g. QuAiL [20] studies how data can be collected efficiently at the physical and link layer within time frames deemed "short" (e.g. 8.192 ms in [20]), while Twofer studies how data collection can be tuned for specific notions of application performance and specific latency budgets with sub-millisecond granularity.

8 CONCLUSION AND OUTLOOK

This paper introduced Twofer, the first wireless embedded network protocol to tune physical- and link-layer operation for latency-bound performance rather than eventual reliability. Twofer's transmissions estimate histograms that characterise distributed data: data is conveyed lossily using frames not more than a few bits in size, nodes transmit concurrently to induce on-the-air quantisation, the system foregoes error correction, and the meaning of transmissions depends on their temporal context. A combination of simulation and full-stack test bed-based experiments reveals that this approach accelerates the collection of noisy data: 25-75% less data needs to be collected to ensure a given accuracy level when collecting, for example, locally differentially private sensor data. The system's approach appears future-proof: its performance advantages increase with the strength of the privacy guarantees that are offered. These results imply that conventional aspects of networked system design, such as link-layer error correction, asymptotic performance analyses, and multi-byte frames hamper the performance of emerging classes of noisy, latency-sensitive wireless sensor network applications. Since Twofer relies on unconventional hardware features and is evaluated with respect to applications that are active research subjects themselves, further efforts must mature the underlying hardware platforms and consider the design of increasingly high-level protocols that perform increasingly complex tasks in high-noise, high-privacy settings.

ACKNOWLEDGMENTS

This work is partially funded by Research Fund KU Leuven, J. Oostvogels' Research Foundation - Flanders fellowship (FWO; 11H7923N), the FWO LOCUSTS project, the Flemish Supercomputer Center, and the EU Horizon OpenSwarm project[4].

REFERENCES

[1] n.d.. Raspberry Pi 4. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed: 2023-06-08.
[2] n.d.. scipy.optimize.nnls. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.nnls.html. Accessed: 2023-06-07.
[3] O Abari, H Rahul, and D Katabi. 2016. Over-the-air function computation in sensor networks. arXiv:1612.02307 [cs.NI]
[4] O Abari, H Rahul, D Katabi, and M Pant. 2015. AirShare: distributed coherent transmission made seamless. In *IEEE Conference on Computer Communications*. 1742–1750.
[5] J Acharya, C. L Canonne, C Freitag, Z Sun, and H Tyagi. 2021. Inference under information constraints III: local privacy constraints. *IEEE Journal on Selected Areas in Information Theory* 2, 1 (2021), 253–267.
[6] J Acharya, Z Sun, and H Zhang. 2019. Hadamard Response: estimating distributions privately, efficiently, and with little communication. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 89)*. PMLR, 1120–1129.
[7] M. S Alvim, N Fernandes, A McIver, and G. H Nunes. 2020. On privacy and accuracy in data releases (Invited Paper). In *31st International Conference on Concurrency Theory (CONCUR 2020)*, Vol. 171. 1:1–1:18.
[8] B Andersson, N Pereira, W Elmenreich, E Tovar, F Pacheco, and N Cruz. 2008. A scalable and efficient approach for obtaining measurements in CAN-based control systems. *IEEE Trans. Ind. Informat.* 4, 2 (2008), 80–91.

[4]This document is issued within the frame and for the purpose of the OpenSwarm project. This project has received funding from the European Union's Horizon Europe Framework under Grant 101093046. Views and opinions expressed are however those of the author(s) only and the European Commission is not responsible for any use that may be made of the information it contains.

[9] R Bassily and A Smith. 2015. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing* (Portland, Oregon, USA) *(STOC '15)*. ACM, 127–135.

[10] G Calafiore, F Dabbene, and R Tempo. 1998. Uniform sample generation in l_p balls for probabilistic robustness analysis. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*, Vol. 3. IEEE, 3335–3340.

[11] E. J Candès and M. B Wakin. 2008. An introduction to compressive sampling. *IEEE Signal Processing Magazine* 25, 2 (March 2008), 21–30.

[12] H Cao, S Liu, R Zhao, and X Xiong. 2020. IFed: a novel federated learning framework for local differential privacy in Power Internet of Things. *International Journal of Distributed Sensor Networks* 16, 5 (2020).

[13] W.-N Chen, P Kairouz, and A Ozgur. 2020. Breaking the communication-privacy-accuracy trilemma. In *Advances in Neural Information Processing Systems*, Vol. 33. 3312–3324.

[14] W.-S Choi, M Tomei, J. R. S Vicarte, P. K Hanumolu, and R Kumar. 2018. Guaranteeing local differential privacy on ultra-low-power systems. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*. 561–574.

[15] O Dabeer and A Karnik. 2006. Signal parameter estimation using 1-bit dithered quantization. *IEEE Transactions on Information Theory* 52, 12 (2006), 5389–5405.

[16] U Erlingsson, V Pihur, and A Korolova. 2014. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 1054–1067.

[17] M Farsi, K Ratcliff, and M Barbosa. 1999. An overview of Controller Area Network. *Computing & Control Engineering Journal* 10, 3 (1999), 113–120.

[18] A Fathalizadeh, V Moghtadaiee, and M Alishahi. 2023. Indoor geo-indistinguishability: adopting differential privacy for indoor location data protection. *IEEE Transactions on Emerging Topics in Computing* (2023), 1–13.

[19] A. F Gad. 2020. PyGAD - Python genetic algorithm! https://pygad.readthedocs.io. Accessed: 2023-11-18.

[20] A Gadre, F Yi, A Rowe, B Iannucci, and S Kumar. 2020. Quick (and dirty) aggregate queries on low-power WANs. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '20)*. 277–288.

[21] M Gallagher. 1991. Proportionality, disproportionality and electoral systems. *Electoral studies* 10, 1 (1991), 33–51.

[22] J Gao, M Tang, T Wang, and B Campbell. 2023. PFed-LDP: a personalized federated local differential privacy framework for IoT sensing data. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems* (Boston, Massachusetts) *(SenSys '22)*. ACM, 835–836.

[23] M Goldenbaum, H Boche, and S Stańczak. 2013. Harnessing interference for analog function computation in wireless sensor networks. *IEEE Transactions on Signal Processing* 61, 20 (2013), 4893–4906.

[24] C Gomez, J Oller, and J Paradells. 2012. Overview and evaluation of Bluetooth Low Energy: an emerging low-power wireless technology. *Sensors* 12, 9 (2012), 11734–11753.

[25] H Hellström, J. M. B. d Silva Jr, M. M Amiri, M Chen, V Fodor, H. V Poor, and C Fischione. 2020. Wireless for machine learning. *arXiv:2008.13492* (2020).

[26] N Hou, X Xia, Y Wang, and Y Zheng. 2023. One shot for all: quick and accurate data aggregation for LPWANs. In *IEEE Conference on Computer Communications*.

[27] 2020. IEEE standard for low-rate wireless networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* (2020).

[28] R Jacob, M Zimmerling, C. A Boano, L Vanbever, and L Thiele. 2021. Designing replicable networking experiments with TriScale. *Journal of Systems Research* 1, 1 (2021).

[29] S Jain and J Guajardo. 2016. Physical layer group key agreement for automotive controller area networks. In *Cryptographic Hardware and Embedded Systems*. 85–105.

[30] S Jain, Q Wang, M. T Arafin, and J Guajardo. 2018. Probing attacks on physical layer key agreement for automotive controller area networks. In *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 7–12.

[31] P Jakimovski, F Becker, S Sigg, H. R Schmidtke, and M Beigl. 2011. Collective communication for dense sensing environments. In *2011 Seventh International Conference on Intelligent Environments*. 157–164.

[32] M Jin, Y He, Y Liu, and X Wang. 2022. Furtively connecting IoT devices with acoustic noise. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 195–207.

[33] P Kairouz, K Bonawitz, and D Ramage. 2016. Discrete distribution estimation under local privacy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*. PMLR, 2436–2444.

[34] P Kairouz, S Oh, and P Viswanath. 2014. Extremal mechanisms for local differential privacy. In *Advances in Neural Information Processing Systems*, Vol. 27.

[35] G Khalaf and G Shukur. 2005. Choosing ridge parameter for regression problems. *Communications in Statistics - Theory and Methods* 34, 5 (2005), 1177–1182.

[36] D Kifer and A Machanavajjhala. 2011. No free lunch in data privacy *(SIGMOD '11)*. ACM, 193–204.

[37] D Kifer and A Machanavajjhala. 2014. Pufferfish: a framework for mathematical privacy definitions. *ACM Trans. Database Syst.* 39, 1, Article 3 (jan 2014), 36 pages.

[38] C Klemenjak, C Kovatsch, M Herold, and W Elmenreich. 2020. A synthetic energy dataset for non-intrusive load monitoring in households. *Scientific Data* 7, 1 (2020), 108.

[39] C. L Lawson and R. J Hanson. 1995. *Solving least squares problems*. SIAM.

[40] D. C Lay, S. R Lay, and J McDonald. 2016. *Linear algebra and its applications*. (5th ed.). Pearson Education.

[41] A Li, J Sun, X Zeng, M Zhang, H Li, and Y Chen. 2021. FedMask: joint computation and communication-efficient personalized federated learning via heterogeneous masking *(SenSys '21)*. ACM, 42–55.

[42] H Li, S Gong, L Lai, Z Han, R. C Qiu, and D Yang. 2012. Efficient and secure wireless communications for advanced metering infrastructure in smart grids. *IEEE Transactions on Smart Grid* 3, 3 (2012), 1540–1551.

[43] LoRa Alliance. 2017. *LoRaWAN specification v1.1*. Technical Report. https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1. Accessed: 2023-06-07.

[44] Y Miao, R Xie, X Li, X Liu, Z Ma, and R. H Deng. 2022. Compressed federated learning based on adaptive local differential privacy *(ACSAC '22)*. 159–170.

[45] T Nuradha and Z Goldfeld. 2022. An information-theoretic characterization of Pufferfish Privacy. In *2022 IEEE International Symposium on Information Theory (ISIT)*. 2005–2010.

[46] J Oostvogels, S Michiels, and D Hughes. 2022. One-Take: gathering distributed sensor data through dominant symbols for fast classification. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 337–349.

[47] J Oostvogels, F Yang, S Michiels, and D Hughes. 2020. Zero-Wire: a deterministic and low-latency wireless bus through symbol-synchronous transmission of optical signals. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20)*. ACM, 164–178.

[48] L Ou, Z Qin, S Liao, T Li, and D Zhang. 2020. Singular spectrum analysis for local differential privacy of classifications in the smart grid. *IEEE Internet of Things Journal* 7, 6 (2020), 5246–5255.

[49] N Pereira, B Andersson, and E Tovar. 2007. WiDom: a dominance protocol for wireless medium access. *IEEE Trans. Ind. Informat.* 3, 2 (2007), 120–130.

[50] M Seif, R Tandon, and M Li. 2020. Wireless federated learning with local differential privacy. In *2020 IEEE International Symposium on Information Theory (ISIT)*. 2604–2609.

[51] C. E Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

[52] S Sigg, P Jakimovski, and M Beigl. 2012. Calculation of functions on the RF-channel for IoT. In *2012 3rd IEEE International Conference on the Internet of Things*. 107–113.

[53] S Song, Y Wang, and K Chaudhuri. 2017. Pufferfish privacy mechanisms for correlated data. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. ACM, 1291–1306.

[54] D Steinmetzer, M Schulz, and M Hollick. 2015. Lockpicking physical layer key exchange: weak adversary models invite the thief. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*.

[55] STMicroelectronics. 2021. STM32G474xB STM32G474xC STM32G474xE. https://www.st.com/resource/en/datasheet/stm32g474cb.pdf. Accessed: 2023-06-10.

[56] Texas Instruments. 2019. AM335x and AMIC110 Sitara processors – technical reference manual. https://www.ti.com/lit/ug/spruh73q/spruh73q.pdf. Accessed: 2023-06-10.

[57] L Theis and N. Y Ahmed. 2022. Algorithms for the communication of samples. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*. PMLR, 21308–21328.

[58] L Vangelista. 2017. Frequency shift chirp modulation: the LoRa modulation. *IEEE Signal Processing Letters* 24, 12 (2017), 1818–1821.

[59] S Wang, L Huang, Y Nie, P Wang, H Xu, and W Yang. 2018. PrivSet: set-valued data analyses with local differential privacy. In *IEEE Conference on Computer Communications*. 1088–1096.

[60] T Wang, J Blocki, N Li, and S Jha. 2017. Locally differentially private protocols for frequency estimation. In *26th USENIX Security Symposium*. 729–745.

[61] T Wang, Z Cao, S Wang, J Wang, L Qi, A Liu, M Xie, and X Li. 2020. Privacy-enhanced data collection based on deep learning for Internet of Vehicles. *IEEE Transactions on Industrial Informatics* 16, 10 (2020), 6663–6672.

[62] T Wang, X Zhang, J Feng, and X Yang. 2020. A comprehensive survey on local differential privacy toward data statistics and analysis. *Sensors* 20, 24 (2020), 7030.

[63] S Werner and J Lundén. 2016. Smart load tracking and reporting for real-time metering in electric power grids. *IEEE Transactions on Smart Grid* 7, 3 (2016), 1723–1731.

[64] X Xiong, S Liu, D Li, Z Cai, and X Niu. 2020. A comprehensive survey on local differential privacy. *Security and Communication Networks* 2020 (2020), 1–29.

[65] R Zamir. 2014. *Lattice coding for signals and networks: a structured coding approach to quantization, modulation, and multiuser information theory*. Cambridge University Press.

[66] A Şahin and R Yang. 2023. A survey on over-the-air computation. *IEEE Communications Surveys & Tutorials* (2023).