Zoom2Net: Constrained Network Telemetry Imputation

Fengchen Gong
Princeton University

Divya Raghunathan
Princeton University

Aarti Gupta
Princeton University

Maria Apostolaki
Princeton University

Abstract

Fine-grained monitoring is crucial for multiple data-driven tasks such as debugging, provisioning, and securing networks. Yet, practical constraints in collecting, extracting, and storing data often force operators to use coarse-grained sampled monitoring, degrading the performance of the various tasks. In this work, we explore the feasibility of leveraging the correlations among coarse-grained time series to impute their fine-grained counterparts in software. We present Zoom2Net, a transformer-based model for network imputation that incorporates domain knowledge through operational and measurement constraints, ensuring that the imputed network telemetry time series are not only realistic but align with existing measurements. This approach enhances the capabilities of current monitoring infrastructures, allowing operators to gain more insights into system behaviors without the need for hardware upgrades. We evaluate Zoom2Net on four diverse datasets (*e.g.,* cloud telemetry and Internet data transfer) and use cases (*e.g.,* bursts analysis and traffic classification). We demonstrate that Zoom2Net consistently achieves high imputation accuracy with a zoom-in factor of up to 100 and performs better on downstream tasks compared to baselines by an average of 38%.

CCS Concepts

• **Networks** → **Network monitoring**.

Keywords

Telemetry, Imputation, Formal Methods, Transformer

ACM Reference Format:
Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2024. Zoom2Net: Constrained Network Telemetry Imputation. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* , 14 pages. https://doi.org/10.1145/3651890.3672225

1 Introduction

Imagine a large datacenter operator tasked with pinpointing the root cause of an instance of packet drops occurring on a network switch. They speculate on several potential issues, such as buffer shortage, bursty traffic, misconfiguration, or potential hardware problems. To identify the actual cause, the operator examines different monitored signals, including active end-to-end latency, packet counts, and queue lengths. However, they quickly realize that the collection interval of these measurements is not fine-grained enough to provide the necessary insights. This is a typical problem as very fine-grained monitoring is hindered by hardware limitations or cost factors. In theory, the operator could request the collection of more fine-grained queue-length data from all devices and wait for the next instances of packet drops to identify the root cause.

As collecting more fine-grained telemetry will be time-consuming and even infeasible for some devices, the operator would, most likely, end up looking back at the existing coarse-grained measurements, trying to piece together likely scenarios in their mind. This involves speculating about concrete scenarios, mentally constructing a finer-grained version of the measured signals, and aligning these speculations with the actual measurements they have. For instance, if there were spikes in end-to-end latency measurements of distinct queues before the spurious drops, the operator might speculate that multiple queues' lengths were consistently high, which allowed the random sampling to 'catch' multiple high values, then those consistently high queues could have filled the switch buffer, leading to drops. This reasoning is feasible due to the underlying correlations between various network signals (*e.g.,* end-to-end delay, queue lengths, drops), which the operator understands and could use to make informed inferences (guesses) about the events that occurred.

While intuitive in this example, conceptualizing various signals and filling the gaps in monitoring generally is extremely challenging even for seasoned operators. The challenge lies in accurately identifying, utilizing, and making sense of the correlations among various signals. Additionally, the sheer breadth of the potential search space adds to the complexity, making both manual and automated reasoning methods unscalable. Still, the scenario raises a question: can we design a system that automatically analyzes multiple correlated, coarse-grained network signals to recover a more detailed picture of the network? Doing so would allow operators to maximize the value of their existing monitoring infrastructure, indirectly improving multiple management tasks with no hardware investment.

We introduce Zoom2Net, a system that imputes fine-grained network monitoring data from multiple coarse-grained ones. This is possible because the various sampled (coarse-grained) time series not only constrain their own imputed versions (imputed signals need to be consistent with the measurements) but also impose constraints on the imputed versions of each other. For instance, coarse-grained queue-length samples constrain the imputed, more fine-grained queue lengths, and these, in turn, are also constrained by the packet counts (as a queue requires a sufficient number of packets to be received to form). While Zoom2Net addresses an inherently under-constrained problem, we observe that there are additional correlations among measured time series that further narrow down potential outputs, often associated with traffic patterns or challenging-to-formulate correlations. Under these correlations, certain scenarios are more likely to occur repeatedly, contributing to a more predictable aspect of the problem. Zoom2Net can train on a small set of fine-grained data generated from an expensive on-demand monitoring tool (or packet trace) and then be used to improve the granularity of always-on monitoring infrastructure.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672225

At first glance, Zoom2Net resembles super-resolution, which recovers high-resolution images from their low-resolution counterparts using deep learning: a concept extensively explored in the literature [20, 36, 47, 48, 53, 54]. Similarly to images where there are correlations within the RGB values and across adjacent pixels allowing the imputation tasks, network signals are often correlated. Some correlations can be expressed as mathematical equations or limits, while others manifest as hidden patterns.

Despite the rich literature, off-the-shelf ML models [14] prove inadequate for Zoom2Net. Network telemetry imputation demands not only realism but also consistency with existing measurements and adherence to known principles, a requirement often neglected by general ML models. Unlike typical super-resolution tasks, which prioritize visually appealing outputs, network imputation necessitates recovered fine-grained time series to *(i)* closely resemble ground truth; and *(ii)* produce the original coarse-grained time series if sampled with the same sampling process. Integrating this kind of domain knowledge into ML models is complex: designed to learn from data, ML models cannot easily ingest traditional knowledge, such as rules and relationships. As a result, there is no standard way of doing so. Moreover, defining the success criteria for network imputation proves challenging. Commonly used metrics like mean-square error (MSE) can penalize outputs that are practically equivalent, such as temporally shifted bursts. Even more complex is the scenario where the same coarse-grained data corresponds to multiple fine-grained possibilities in the training set (ambiguity), potentially hindering the ML model's effective training.

Instead of solely relying on data for training, Zoom2Net incorporates different sources of knowledge to generate not only realistic but accurate fine-grained time series using two specific methods. First, utilizing a transformer model [46] at its core, Zoom2Net employs a knowledge-augmented loss function that embeds both *operational* and *measurement constraints*, guiding the transformer to learn both correlations and properties. Second, Zoom2Net enforces the constraints that the transformer failed to satisfy by using a *constraint enforcement module* to correct its output post-imputation.

To manage the inherent ambiguities in data, which tend to increase as the zoom-in factor (the ratio between coarse and fine granularity) becomes more pronounced, we have modified our strategy. Instead of aiming for precise, point-to-point accuracy regardless of the zoom-in factor, we train Zoom2Net to generate outputs that are, at least, plausible and consistent with measurements. This change not only improves the efficiency of the training process but also leads to more practical and useful outputs in real-world applications.

We evaluate Zoom2Net across synthetic and real-world datasets and compare its performance to both statistics and learning-based methods. We find Zoom2Net achieves: *(i)* consistently high imputation accuracy with a zoom-in factor of up to 100; *(ii)* 38% better performance in downstream tasks compared to baselines; and *(iii)* demonstrates its ability to apply learned correlations to scenarios unseen during training.

This work builds on our previous position paper [27] that outlined the potential of combining formal methods with machine learning for networking. This work does not raise any ethical issues.

2 Motivation and limitations of existing work

In this section, we discuss motivating scenarios where fine-grained telemetry is required after the fact. Next, we explain why existing solutions are unlikely to help.

2.1 Use cases and requirements

Post-Mortem Analysis. In the aftermath of a volumetric attack, network operators leverage monitoring to analyze the event. This data helps in revealing the attack's impact across the network, pinpointing vulnerable zones, and assessing the defense mechanisms' performance under real stress. Insights gained from this analysis are crucial for strengthening the network's resilience against future attacks. They can guide adjustments in security policies, firewall rules, and even the deployment of advanced intrusion detection systems that can better handle sudden surges in traffic. While vital, the collection and storage of every potential signal at the finest level of granularity continuously is impractical. Thus, having a way to zoom into a particular signal that is being monitored after the fact is extremely useful.

Intuitively, a learning approach is unlikely to generate novel data patterns corresponding to a new attack. Yet, in the context of network behavior, individual monitoring signals may not deviate significantly from normal patterns. In other words, a novel or rare attack could manifest as an escalation in the frequency of regular incidents, or it may be the result of a specific sequence and combination of routine occurrences that serve the attacker's objectives rather than a set of anomalous or unforeseen behaviors on each signal. Hence, a learning approach can, in fact, zoom into the incident. For example, consider a shrew attack [33] consisting of periodic bursts that disrupt TCP. Such malicious behavior can only be revealed by *fine-grained* queue lengths, which can be recovered as individual bursts are regular incidents.

Diagnostic Troubleshooting. To identify the root cause of an alert or client's complaint, network operators need to check various monitored signals potentially on multiple devices and try to identify anomalies or deviations from normal operations. Note that these anomalies could only be visible when the monitoring interval is adequately small, while the exact signals of interest are not necessarily known. For instance, dropped packets in a flow caused by bursts can only be observed with millisecond granularity measurements on all devices on each of the paths through which a flow could have been forwarded. Thus, fine-grained and general monitoring is critical for prompt and precise troubleshooting and, thus, faster resolution of network issues. We evaluate this use case in §6.2.

Network resource planning. To plan for capacity increases, on-device memory provisioning, topology design, peering agreements, and many other resources, an operator needs detailed insights into the network's usage patterns and demands over long periods of time. Fine-grained monitoring of various signals would help to avoid both under-provisioning, which can lead to congestion and performance issues, and over-provisioning, which can be costly and inefficient. On the one hand, collecting and storing all possible monitoring data for long periods to satisfy future operators' demands is wasteful, given the data volume. On the other hand, predicting signals of interest and their required granularity is impractical given the technological advancements and evolving application demands. For example, an

operator might need to decide on the buffer size to provision on network devices, for which they would need to see historical data on the aggregate occupancy of the buffer in quantity (percentage of buffer occupied) and in quality (stably long queues or bursty traffic) [6, 7]. Yet, the data might have been collected at a time when buffer size was not a limiting resource, hence not a signal of interest, or at a time when the latency of milliseconds was acceptable; hence, higher granularity seemed unnecessary. In such cases, having a way to zoom into signals would be a great software alternative. We evaluate this use case in §6.2, and §6.3.

Dataset imputation. To protect their networks from attacks, load balance traffic, or debug issues, operators often use machine learning. To train their models, the operators will turn the initial packet traces into a series of features [21]. Oftentimes, the raw traces are deleted after the features have been calculated to reduce the risks of a data breach or to save space. The features are stored and/or shared. While convenient, this solution prevents the operators from going back to the raw trace and calculating new features that they conjecture might be useful for their operation. Of course, they might be able to collect new traces if they own the infrastructure, but that would be time-consuming and might hurt performance (e.g., if the previous dataset happened to contain attacks or infrequent scenarios). In other words, many datasets today can be seen as coarse-grained representations of raw data, retrieving which would allow for easier experimentation. In this case, having a way to 'zoom into' features to see the raw dataset would be very useful. While this use case might deviate from our core goals, we see encouraging results in §6.4; after all, feature engineering on network traces is a form of sampling.

2.2 Limitations of existing works

In various use cases, a recurring theme is the need for techniques that gather telemetry data to meet three specific requirements: *(i)* **generality**: the technique should be applicable to many different types of telemetry. This is critical as operators are often uncertain about signals of interest *a priori*; *(ii)* **fine granularity**: telemetry should be collected frequently enough to facilitate the detection of short-timescale changes (*e.g.,* bursts); and *(iii)* **cost efficiency**: the processing, memory and storage needed for telemetry should be low and amenable to be deployed on commodity hardware. Next, we summarize why existing monitoring solutions failed to fulfill these requirements.

Traffic mirroring: Mirroring traffic from routers to collectors [42, 45] using specialized hardware can satisfy granularity and generality as they may allow an operator to run custom queries at any granularity. However, this does not meet cost efficiency as traffic volumes increase; *e.g.,* mirroring requires significant bandwidth and processing, and custom hardware may become too expensive.

Sampling & Network Tomography: Packet sampling approaches (*e.g.,* SNMP, sFlow) can reduce costs. Unfortunately, polling counters is resource intensive and hence typically done at a coarse timescale, *e.g.,* minutes. Thus, it is not fine-grained and cannot be used to detect microbursts [44]. Network tomography can make more out of the collected data, not by increasing the granularity but by inferring unmeasured metrics (*e.g.,* latencies of sub-paths). Both traditional and more advanced network tomography [24] only work on linear relations between signals and known network topology, lacking generalization.

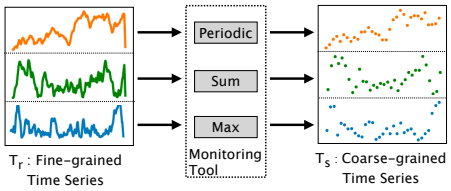


Figure 1: Fine-grained signals T_r of a networked system are sampled by monitoring tools, resulting in coarse-grained time series T_s available to operators. In practice, T_r is only available for training and is collected over a short period using specialized hardware or traffic mirroring.

Programmable switch & eBPF: Recent developments in programmable switches have enabled different methods of collecting network telemetry. In-band network telemetry (INT) is a technique that embeds fine-grained and accurate telemetry in each packet. While effective, INT generates a substantial amount of data, leading to high memory and bandwidth usage, thus, is not cost-efficient. Lighter versions of INT [10, 34, 41] still require homogeneous programmable hardware and generate a large amount of data. Customized algorithms such as sketches offer a cost-efficient approach to generating fine-grained telemetry. However, these algorithms are typically designed for specific tasks at design time (*e.g.,* heavy-hitter detection [16, 38], latency monitoring [8, 11, 43]) or identifying culprits [15, 37], lack generalization across tasks, and often generate huge amounts of data that are hard to extract from the device and store. Even highly optimized solutions using eBPF often fall into scalability walls. For instance, to reduce its memory and CPU usage, Millisampler [25] can only monitor and collect fine-grained telemetry for a short period (*e.g.,* 20 seconds), potentially missing the most critical events.

3 Overview

Traditional monitoring techniques fall short in one or more of the requirements (cost-efficiency, fine granularity, and generality) in §2.2. Our overarching goal is to develop a framework that can fulfill these requirements in software through network telemetry imputation. We formulate the problem and then explain the challenges and insights that drove our design.

3.1 Network imputation

Consider a physical networked system that is described by a set of fine-grained *raw* time series $T_r = \{T_r^1, T_r^2, ...T_r^n\}$. A monitoring infrastructure senses the physical system using a known sampling/coarsening function S, and outputs $T_s^i = S(T_r^i)$. For instance, S could sample a single value out of every N values in T_r^i, or it could output the average/max/mean for every window. In total, the output of the sampling function is a set of coarse-grained time series $T_s = \{T_s^1, T_s^2, ...T_s^n\}$. We illustrate this in Fig. 1. We aim to recover (impute) an approximate T_r^i: $\hat{T}_r^i$, which, if given as input to the various management tasks, can improve their performance compared to if their input was from T_s^i. This process is illustrated in Fig. 2. T_r is

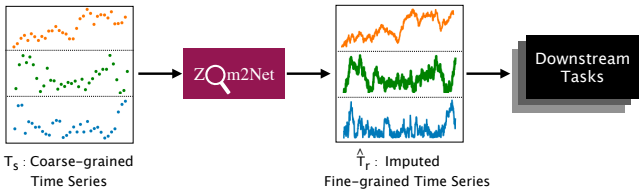


Figure 2: Zoom2Net takes a set of coarse-grained time series T_s as input and outputs imputed fine-grained time series $\hat{T}_r$ which is fed to multiple downstream tasks.

sampled above the Nyquist frequency; thus, there is information loss that prevents its trivial recovery. We assume there is available a dataset of fine-grained time series from the network of interest. This can be practically generated by running for a short period: *(i)* an unscalable monitoring tool that cannot be always-on because of CPU/storage usage, or *(ii)* an advanced hardware device only temporarily plugged into the network or *(iii)* a simple tab/mirror of traffic. We do not assume the T_s to be perfectly aligned in time nor monitored on the same granularity.

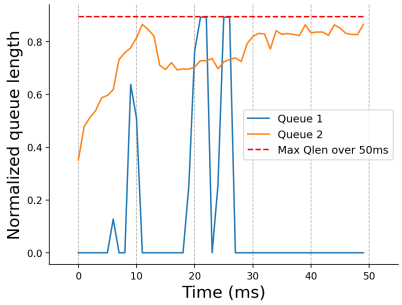
Non goals. Our system works offline for model inference and data analysis. While this would prevent real-time tasks, our system could already be used to improve debugging, provisioning, and attack analysis (including all the use cases in §2.1).

3.2 Challenges and Insights

We describe the challenges and insights, through a running example of a network switch.[1] Monitoring queue lengths at fine granularity could benefit network management tasks, including finding the root cause of packet drops and network planning. For this example, we consider fine-grained queue lengths to be of 1ms granularity and coarse-grained 50ms.

Challenge 1: A single coarse-grained time series is often ambiguous, making imputation impossible. The network imputation problem is, by nature, under-constrained: multiple versions of fine-grained time series, when sampled *e.g.,* below the Nyquist rate, can produce identical coarse-grained time series. As a result, it is fundamentally difficult (if not impossible) to reconstruct the correct fine-grained version.

We show an example in Fig. 3a, where we plot the queue length of two queues at fine-grained intervals (at 1ms). Queue 1 has multiple short bursts while queue 2 has a constantly high queue length. While the queues display distinct patterns, applying reasonable monitoring tools such as LANZ [1], which outputs the maximum queue length at every interval (50ms in this case), results in identical coarse-grained time series. This is problematic for an operator observing only the coarse-grained version, as these different queue patterns necessitate varied approaches for resolution. For instance, a consistently high queue, such as queue 2 (orange), may require adjustments in congestion control or the adoption of an Active Queue Management (AQM) system. Conversely, a bursty queue pattern, such as queue



(a) Two fine-grained queue length time series with distinct behaviors at 1ms.

	Max Qlen	Packet Drop	Packet Sent
Queue 1	0.895	0.05	0.74
Queue 2	0.87	0.69	0.98

(b) Signals for two queues sampled at 50ms.

Figure 3: Two distinct fine-grained queue length behaviors result in the same sampled maximum queue length. But they can be distinguished from different sampled packet drops and sent counts.

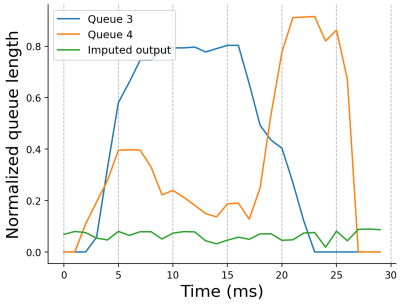
1 (blue), suggests the presence of a bursty application, calling for a strategy like pacing for effective management.

Insight: Leveraging multiple time series can often resolve the ambiguity. Relying solely on a single coarse-grained queue monitoring time series may not suffice to impute fine-grained queue length. Still, integrating additional time series can provide the necessary clarity. Specifically, the two queues in Fig. 3a naturally exhibit significantly different patterns in terms of packet counts and drop counts which are also typically monitored *e.g.,* by SNMP [22]. We can observe this in the normalized Packet drop and Packet sent columns of Table 3b. Thus, by examining SNMP-like data covering the same time interval, we can discern distinct traffic profiles for these two queues. In other words, because queue lengths, packet counts and drop counts time series are correlated, we can achieve higher accuracy in imputing fine-grained queue length if we use all three coarse-grained signals compared to using only maximum queue length.

This situation is typical in networking, where various monitored time series often display correlations. For example, in data centers, traffic rates for servers within the same Top-of-Rack (ToR) are correlated because they share an uplink (thus, these servers cannot be concurrently sending data at full capacity in oversubscribed networks). Similarly, end-host traffic volume correlates with congestion window size and Round Trip Time (RTT), which denote current network conditions.

Leveraging these correlations is a great opportunity for solving the imputation problem, but they are not trivial to capture. Recent advancements in generative models [12, 13, 31, 39, 52] offer a promising solution. Transformers, as sequence-to-sequence models, excel at learning correlations across lengthy sequences efficiently via the attention mechanism [46]. Their capacity to generalize effectively and capture diverse contexts allows them to grasp shared

[1] Data for this example is from an ns-3 simulation which we describe in §6.2.



(a) Two fine-grained (per 1ms) queue lengths time series with the imputation result of a plain transformer model.

	Max Qlen	Packet Drop	Packet Sent
Queue 3	0.80	0.06	0.60
Queue 4	0.92	0.086	0.64

(b) Coarse-grained measurements (per 50ms) related to the two queues.

Figure 4: Two distinct fine-grained queue length time series result in almost identical coarse-grained signals. A transformer model trained with MSE, having seen both (and more), would generate an average (green line), obfuscating the burst.

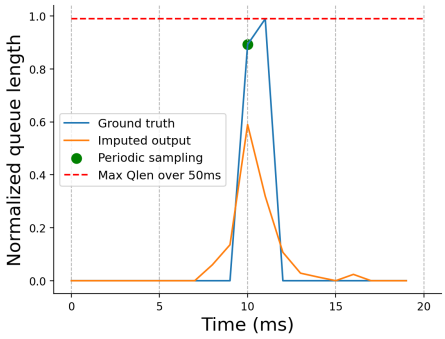


Figure 5: Ground truth fine-grained queue length at 1ms (blue) and imputed fine-grained queue length from a plain transformer at 1ms (orange). A plain transformer catches trends but outputs results inconsistent with maximum and periodic samples.

dynamics. This flexibility and efficiency have already established their popularity in the field of networking [18, 28, 35].

Challenge 2: Ambiguity might remain even when combining multiple coarse-grained time series. While some scenarios can be recovered correctly by leveraging multiple correlated coarse-grained time series, that is not always the case. Indeed, as we decrease the granularity of the coarse data (increase the zoom-in factor), there will be more clusters of pairs (T_r, T_s) with identical coarse-grained series (T_s) but distinct fine-grained counterparts (T_r). Fig. 4a illustrates such a case with two queues experiencing a burst at different times, hence having very different fine-grained queue length time series. Unlike the previous case (Fig. 3a), though, all coarse-grained time series T_s, namely the maximum queue length, the packet drop counts, and the sent packet count, are almost identical, as we observe in Table 4b. Such cases, which we call coarse-grained collisions, are detrimental to both training and inference. First, a model trained on a dataset with multiple coarse-grained collisions would take longer to converge and might be unstable. At a high level, this is the same problem as having a sample with multiple labels [50, 51]. Second, the output of an ML model in coarse-grained collisions might be useless. To illustrate this problem, we train a transformer with simple MSE loss and observe its behavior in imputing queue 3. The transformer ends up producing the average of all scenarios with the same coarse-grained input, as we illustrate in Fig. 4a in green, which would be completely useless (as it hides the burst itself).

Insight: A plausible (rather than perfect) imputation output is a more attainable and still useful goal (§4.4). We observe that oftentimes in coarse-grained collisions *i.e.,* when all correlated coarse-grained time series are similar, the corresponding ground-truth fine-grained time series (albeit seemingly different) correspond to functionally equivalent scenarios.In our example in Fig. 4a, for instance, there is a burst of similar duration and rate that is shifted in time. In such an instance, a system generating any of these versions would generally meet the expectations of the network operator interested in burst detection. On the contrary, an average of multiple instances(as we illustrate in Fig. 4a) would not be acceptable. Even if the colliding instances do not represent functionally equivalent scenarios, outputting the correct fine-grained version of one of them is more useful than outputting an average of all colliding instances. Ultimately, our primary goal was to automate the thought process of a highly skilled operator, who would typically attempt to align the coarse-grained data with any instance they have seen before and matches. Nonetheless, grouping colliding scenarios can be complex.

Challenge 3: ML does not provide guarantees. The most significant downside of any ML in the context of telemetry imputation is that the output lacks correctness guarantees. For example, in Fig. 5, we observe that the imputed time series (orange line) of a queue's length (blue line) generated by a transformer with MSE loss (plain transformer) is not consistent with the measurements: the transformer did not impute a queue length that is as high as the (known) max queue length (red dashed line) of the interval, and the output at 10ms is not consistent with the corresponding periodic sample, although they are part of the transformer's input. It may seem surprising that the plain model does not "realize" the connection between the provided max/periodic queue lengths and the ground-truth (fine-grained) queue lengths. However, this issue arises due to the inherent challenge of predicting large values when the input data is predominately skewed towards smaller values. To make matters worse, the output of the plain model also violates switch-specific constraints. For example, the total number of packets that would need to have been dequeued for the imputed queue to be formed exceeded the SNMP count.

Insight: Incorporate knowledge and enforce consistency with measurements (§4.2, §4.3). Instead of solely relying on data to train our model, we leverage knowledge. Concretely, we observe that there are often correlations across monitored time series that can be formalized. For instance, given a queue length time series, we can calculate the number of packets dequeued, which should not exceed the number of packets sent out. Further, we know that the system's output should be consistent *i.e.,* produce the coarse-grained time

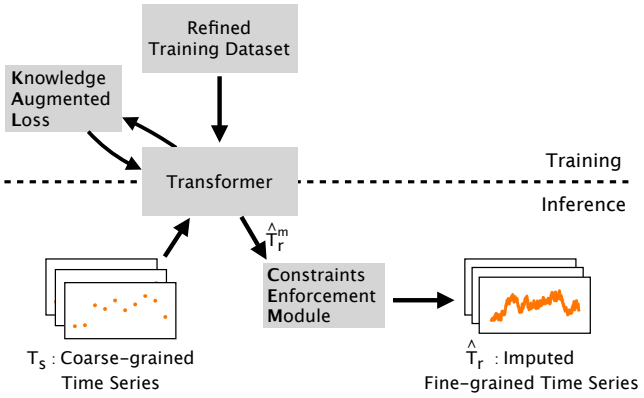


Figure 6: During training, a transformer model takes refined training dataset and trains with Knowledge Augmented Loss function. During inference, the model takes in coarse-grained time series T_s and outputs $\hat{T_r^m}$ which is then corrected by Constraints Enforcement Module. The result $\hat{T_r}$ is used for downstream tasks.

series if sampled with the corresponding operator. We can leverage these connections to guide the output towards more accurate results. Inspired by Physics and the extensive work in Physics-Informed-Neural-Networks [19, 32] and the imaging literature [9], we incorporate this knowledge during training on the loss function and during inference through a consistency enforcement module. By embedding domain knowledge, we increase not only the model's accuracy, but also its reliability *i.e.,* the operator can have more confidence that the result produced is plausible.

4 Zoom2Net Design

Driven by these insights, we design Zoom2Net, which we illustrate in Fig 6. We start with discussing the formulation of knowledge (§4.1), which we incorporate into the loss function, forming the Knowledge Augmented Loss (KAL) function (§4.2), and into a post-imputation Constraints Enforcement Module (CEM) for output correction (§4.3). Finally, we discuss ways of refining our training dataset to reduce the effect of coarse-grained collisions (§4.4).

4.1 Knowledge formulation

In the context of Zoom2Net, we categorize knowledge into two types: measurement knowledge and operational knowledge.

Measurement knowledge demands that applying monitoring tools (*e.g.,* max queue length, packet counts) on the imputed fine-grained time series output should result in coarse-grained measurements. Inspired by formal methods, we articulate such knowledge as equality constraints $\Phi(\hat{T_r}, T_s)$.

$$\Phi(\hat{T_r}, T_s) = 0 \qquad (C_{equal})$$
$$where \quad \Phi(\hat{T_r}, T_s) = T_s - S(\hat{T_r})$$

Here, S represents the sampling/coarsening function. Importantly, such constraints are extremely easy to identify because they are the result of monitoring.

The operational knowledge captures the correlations between different signals. For instance, the count of enqueued packets should not exceed the count of sent packets. These relationships are expressed as inequality constraints $\Psi(\hat{T_r}, T_s)$.

$$\Psi(\hat{T_r}, T_s) \leq 0 \qquad (C_{inequal})$$

There are cases where correlations include comparison and logical operators (*e.g.,* $\leq, \vee$). To integrate such constraints into our system, we transform the operators into expressions that output $True$ or $False$. For example, to test if the constraint $a < b$ is satisfied, we formulate it as $step(a - b)$ with a step function.

Identifying and formulating measurement constraints is straightforward, as operators are typically familiar with their monitoring functions/tools. Formulating operational constraints might require more domain knowledge, but identifying them can be automated by running pure correlation tests *e.g.,* Pearson's Correlation Coefficient among the available signals.

Zoom2Net does not necessitate time series to be perfectly aligned. Indeed, as the transformer learns directly from data it can catch correlations from time series that are not perfectly synchronized (*e.g.,* due to unsynchronized clocks or because of different granularity intervals). If there is an unknown time shift, it is (in theory) possible that the operational constraints are violated in the ground truth (thus, it is better not to be used). Yet, in practice, we don't see this because operational constraints are typically loose. Measurement constraints are not affected.

4.2 Knowledge Augmented Loss (KAL)

We designed our loss function to enable the model to learn data correlations and distributions, while also guiding it to satisfy the constraints derived from our knowledge.

4.2.1 Loss metrics. Loss functions that minimize point-wise distance, such as MSE, are designed to make the output closely resemble the ground truth, allowing the model to learn data correlations and patterns. However, we noted that MSE encourages the model to find averages among plausible solutions. This becomes particularly problematic when the data is predominately skewed towards smaller values, resulting in overly smooth outputs that struggle to capture bursts [36]. To address the challenge of preserving correlations while overcoming the tendency towards averaged behaviors, we introduce Earth Mover's Distance (EMD) as another term in the loss function. EMD measures the minimum cost required to transport the mass of one distribution to match another. By incorporating EMD, we aim to encourage the model output not only to closely match target values but also to mirror the structure and uncertainty inherent in the target distribution. By minimizing both MSE and EMD, our loss function guides the model to adapt to the data's characteristics. This combined loss function is expressed as:

$$L_{combine} = MSE(\hat{T_r^m}, T_r) + \lambda\, EMD(\hat{T_r^m}, T_r)$$

where $\hat{T_r^m}$ is model output and T_r is target value, λ is a hyperparameter to balance the two loss terms.

4.2.2 Knowledge constraints satisfaction. Given the equality and inequality constraints defined in §4.1, the challenge is how to inform the transformer of the knowledge we have. To address this limitation, our goal is to solve for a set of transformer parameters

minimizing the loss $L_{combine}$ over the training dataset while also satisfying all the knowledge-based constraints. That is, we aim to solve the optimization problem:

$$\min\ L_{combine} \text{ s.t. } \Phi_k(\hat{T_r^m}, T_s) = 0, \quad k \in \{1, ..., K\} \qquad (1)$$
$$\Psi_h(\hat{T_r^m}, T_s) \leq 0, \quad h \in \{1, ..., H\}$$

where K and H are the number of equality and inequality constraints, respectively. To enable the model to learn and adhere to the specified constraints, we adopt the augmented Lagrangian method, inspired by [19]. This method involves introducing penalty terms into the objective function to account for constraint violations. To do so, we further convert the constraints to differentiable terms. For instance, we leverage hyperbolic functions to represent a smoothed step function.

For each constraint in (1), we define a separate Lagrange variable. We define a variable $\lambda_{k,i}^{eq}$ for each equality constraint Φ_k evaluated at each training data $(\hat{T_{r_i}}, T_{r_i})$, and similarly $\lambda_{h,i}^{ineq}$ at each point $(\hat{T_{r_i}}, T_{r_i})$ for all the inequality constraints Ψ_h. The augmented Lagrangian loss function is then given by:

$$\begin{aligned} L_{aug} = L_{combine}(\hat{T_r^m}, T_r) &+ \sum_{\substack{i\in N\\ k\in K}} \mu\Phi_k(\hat{T_{r_i}^m}, T_{s_i})^2 \\ &+ \sum_{\substack{i\in N\\ k\in K}} \lambda_{k,i}^{eq}\Phi_k(\hat{T_{r_i}^m}, T_{s_i}) + \sum_{\substack{i\in N\\ h\in H}} \lambda_{h,i}^{ineq}\Psi_h(\hat{T_{r_i}^m}, T_{s_i},) \\ &+ \sum_{\substack{i\in N\\ h\in H}} \mu[\lambda_{h,i}^{ineq} > 0 \vee \Psi_h > 0]\Psi_h(\hat{T_{r_i}^m}, T_{s_i})^2 \end{aligned}$$

where N is the training dataset size and μ is penalty coefficient. We initialize μ to be 1e−3 and λ to be 0. During each iteration of training, we minimize L_{aug} via gradient descent while keeping the values of μ and λ fixed. After the transformer model has converged, we update μ and λ according to the update rules:

$$\begin{aligned} &\mu \leftarrow \mu * \mu_{mult} \\ &\lambda_{k,i}^{eq} \leftarrow \lambda_{k,i}^{eq} + 2 * \mu * \Phi_k(\hat{T_{r_i}}, T_{s_i}) \\ &\lambda_{h,i}^{ineq} \leftarrow (\lambda_{h,i}^{ineq} + 2 * \mu * \Psi_h(\hat{T_{r_i}}, T_{s_i}))_+ \\ &where\ x_+ = max\{0,\ x\} \end{aligned}$$

where μ_{mult} is a hyperparameter of value 1.5. Then, we start another round of training on the transformer model with the updated Lagrange variables. This process is repeated until the constraint violations reach a saturation point and stop decreasing. In each iteration, the Lagrange multipliers λ are updated by incrementing them based on the violations of the corresponding output data multiplied by μ. The importance of a violation in the loss function increases as the violation magnitude becomes higher, requiring more effective minimization. Training with these penalty terms enables the model to learn the consistency between the input and output, enforced by the constraints.

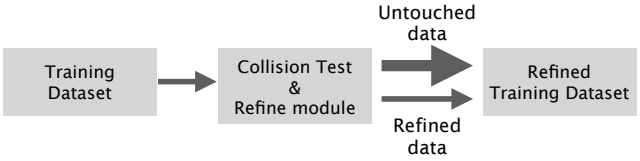


Figure 7: Training dataset goes through a collision test where a small portion of data with close coarse-grained input but distinct fine-grained output is consolidated into one class while the majority remains untouched, forming a refined training dataset.

4.3 Constraint Enforcement Module (CEM)

While the incorporation of constraints in the loss function improves the imputation accuracy, it still provides no guarantee that the constraints will be satisfied. Thus, we introduce the Constraint Enforcement Module (CEM) which aims at correcting the output of the transformer (*i.e.,* forces it to satisfy the specified constraints) while changing it as little as possible. CEM uses the Integer Linear Programming (ILP) solver Gurobi [2] to correct the output of the ML model according to the constraints $(C_{equal}, C_{inequal})$. We use variables $\hat{T_r}[t]$ to denote the corrected output at each time step. To ensure that the corrected time series remains close to the ML model's output, we use the following objective that minimizes the total difference between the corrected and original values, ignoring the time steps in which the data is sampled.

$$\min \sum_{t=0,\ t\notin T_{samples}}^{T-1} |\hat{T_r}[t] - \hat{T_r^m}[t]|$$

4.4 Target refinement

As we discussed in §3.2, certain scenarios involve the same coarse-grained input T_s occurring multiple times in the training dataset, each time associated with a distinct fine-grained target. This one-to-multiple mapping poses challenges for the training convergence of the transformer and risks the usefulness of the result. To address this challenge, we designed a target refinement module shown in Fig 7. The module acts solely on training data and is composed of a collision test and a refinement mechanism. We first discuss how to refine the distinct targets for the same coarse-grained input, and then delve into how to identify the training data that needs to be refined automatically.

First, we observe that although the distinct target values associated with the same input exhibit variations in values, they share the same trends and patterns. If presented with only a single target, the transformer model should learn these shared patterns effectively. Therefore, we aim to provide the model with the same set of plausible targets for the same input in the training set. We use a set of targets instead of one because all target values are valid and equally plausible. To achieve this, we consolidate the different targets associated with the same input T_s into a class T_r^{class}. We define the loss L_{class} to be the minimum difference between the transformer output $\hat{T_r^m}$ and each target in the class.

$$L_{class} = \min_{\forall T_r \in T_r^{class}} L_{combine}(\hat{T_r^m}, T_r)$$

This encourages the transformer to match its output to the closest target and backpropagate the difference. Even when the same input occurs multiple times, the transformer matches the same target and effectively learns the patterns.

A significant challenge within this approach is defining what constitutes a class that encapsulates data with *same* inputs but *different* outputs. Inputs are rarely numerically identical, and determining the proximity of inputs that should result in the same output is challenging due to the complexity of transformer models with thousands of parameters. To address this challenge, we train a basic transformer using the raw training dataset and $L_{combine}$ as the loss function. After training the basic transformer, instances with close imputed outputs indicate that the transformer cannot distinguish the difference in their inputs, implying that the inputs are close enough. Consequently, we identify inputs in the training dataset as part of the same class if: *(i)* their basic transformer outputs are close; and *(ii)* their fine-grained targets are far apart.

5 Implementation

We implement Zoom2Net using Python 3.8 and Pytorch 2.0. We train the transformer model on an Nvidia Tesla T4-16GB GPU. The model architecture includes a one-layer transformer encoder followed by a linear layer. We choose not to include a decoder in the architecture because the output is conditioned on the rest of the data in the time series, eliminating the need for a masking mechanism. In contrast, GPT (Generative Pre-trained Transformer) models, for example, require a decoder because each token is conditioned on the previous tokens. We utilize 4 parallel heads to attend to time series from different representation subspaces simultaneously. The loss function is optimized using the Adam optimizer. The learning rate starts from 1×10^{-4} and decays by a factor of 0.1 after the model stops improving for 10 epochs. In our testbed, training each use case averages 20 minutes facilitated by the transformer's parallel processing capabilities.

For each dataset used for training and testing, we use the original data as fine-grained ground truth and downsample it to coarse-grained input. The downsampling methods and zoom-in factors depend on the specific real-world use case. The features include correlated telemetry collected together. When the downsampling frequency is periodic, we exclude timestamps from features. In cases where measurements are not periodic, we replace real timestamps with relative timestamps to help transformers capture temporal correlations. We define the length of fine-grained output as window size, and the length of coarse-grained input as $\frac{window\ size}{zoom-in\ factor}$. The window size varies across different use cases based on temporal correlations. We normalize the dataset using the min-max method, training our model on 80% of the samples and evaluating it on the remaining 20%.

When running the ILP solver Gurobi, we parallelize coarse-grained intervals across different processes on multiple cores to accelerate the solving process. For example, solving a 1000ms time window with a zoom-in factor of 50 initiates 20 processes. This parallelization is possible because the constraints are defined independently for each interval.

6 Evaluation

We evaluate Zoom2Net across different case studies, using synthetic and real-world data, and we compare it with state-of-the-art approaches. Our evaluation aims to answer the following key questions on both imputation accuracy and downstream task accuracy:

(Q1) How does Zoom2Net perform compared to directly using coarse-grained data?

(Q2) How does Zoom2Net perform against statistical baselines and state-of-the-art time-series imputation models?

(Q3) How does the performance of Zoom2Net change when we increase the zoom-in factor?

We find that Zoom2Net performs up to 5 times better in downstream tasks compared to using coarse-grained data directly. Zoom2Net outperforms baselines in imputation accuracy by 33-53%, with a similar MSE achieved. It improves downstream tasks by an average of 38% compared to baselines. Zoom2Net's performance in imputation accuracy and downstream tasks degrades by an average of 0.4% and 7% with increasing the zoom-in factor by 50.

6.1 Methodology

We compare Zoom2Net against statistical and ML methods.

Pure coarse-grained data: To evaluate the scenario in which the coarse-grained data is used directly, we still need to convert the time series to the appropriate size first. Hence, we use a statistical method, namely IterativeImputer [40], that retains the periodic samples and models missing values as linear functions of other features iteratively. To incorporate measurements such as the maximum value, we place them at the midpoint of each interval.

K-nearest neighbors (KNN): KNN is a straightforward yet effective technique that has been used for image super-resolution tasks. For a given coarse-grained data, KNN identifies the nearest K training data inputs and calculates the average of the K labels as the output. The choice of K is determined through experimentation to yield optimal performance.

Plain transformer: This is a transformer model trained using MSE (without our improvements *e.g.,* the knowledge incorporation, or the refinement step).

Brits [14]: Brits employs bidirectional recurrent neural networks for imputing missing values in time series data. To adapt Brits to our settings, we incorporate sampled values such as sum and max by placing them at the end of the time interval and use Brits to impute values between periodic samples.

Metrics. We evaluate Zoom2Net and our baselines by their imputation accuracy and their performance in downstream tasks. To quantify imputation accuracy, we calculate autocorrelation, distance, and distribution differences between imputed time series and the ground truth. Next, we evaluate the quality of imputation by comparing the performance of downstream tasks using the imputed results as input against using the ground truth. For each of the metrics, we report average $relative\_error = \frac{|t - t_{real}|}{t_{real}}$ over the testing dataset, where t_{real} is the ground truth of a metric and t is the measured value. Because the errors have very different scales over different

methods, we normalize the relative errors of each metric to [0.1, 0.9] for better visualization.

Case studies and goals. We consider three case studies to show Zoom2Net capabilities. For each of these, we first explain an example scenario in which operators have certain downstream tasks in mind. Next, we explain how we use an existing or synthetic dataset to evaluate Zoom2Net in this scenario. While the dataset we use contains fine-grained time series, we treat this as ground truth; thus, we assume it is not available to the operator to use directly. This is realistic as it is effectively equivalent to collecting very fine-grained data for a very short period of time to train on. The downstream tests run on input that is calculated by Zoom2Net (or by other baselines). The input of Zoom2Net (and of the other baselines) is the coarse-grained version of each time series in the dataset.

6.2 Case study 1: ToR burstiness in a Cloud

Consider an operator of a large data center who has to run a set of downstream tasks, e.g., deciding how much on-chip buffer to provision to network switches, or detecting adversarial traffic patterns. To inform these tasks, the operator needs to learn about burst properties in queue lengths, specifically burst position, height, frequency, interarrival distance, duration, and volume. Accurate analysis requires fine-grained switch queue length measurements at the millisecond level. Alternatively, they can use link utilization measurements as a proxy for queue length collected at data center RTT granularity, *i.e.,* 1ms, as demonstrated by researchers at Meta [25]. We test both cases using a synthetic dataset and a dataset released by Meta.

Synthetic Dataset. We generate a dataset using ns-3 simulator [4], simulating a leaf-spine topology as described in [3, 6]. The switches in the simulation adhere to the features of Broadcom TridentII [5] and are configured with Dynamic Thresholds [17] as buffer management scheme. The generated traffic follows web search and incast traffic patterns, incorporating various settings for traffic load, burst size, burst frequency, and congestion control algorithms (*e.g.,* DCTCP and Cubic). During simulation, we collect fine-grained time series (ground truth), including queue lengths, per-port packet, and drop counts every 1 ms. We generate coarse-grained time series by sampling the fine-grained ones at 50ms granularity, mimicking the following monitoring tools: *i.e., (i)* LANZ [1], which provides the per-queue maximum length within each interval, *(ii)* SNMP [22], which provides per-port counts of packets sent and dropped every interval; and *(iii)* periodic sampling. Our training dataset contains 8,000 data points. **Imputation goal:** Zoom2Net takes maximum, periodic sampled queue length, packets dropped, and packets sent count at *50ms* granularity and produces *1ms* fine-grained queue lengths. We then identify bursts using a specific method in [49] and calculate the relative errors of burst timestamps, height, frequency, interarrival time, duration and volume between the ground truth and Zoom2Net output.

Meta Dataset. We use a public dataset from Meta [25] which contains link utilization, retransmission traffic, in-congestion traffic, and connection counts at a fine-grained resolution of 1ms. We aggregate them at intervals of 50ms to create coarse-grained data. In total, we use a training set of 20,000 data points. **Imputation goal:** Zoom2Net takes the aggregated measurements at *50ms* granularity and produces *1ms* fine-grained link utilization. We identify bursts and their characteristics using the method described in the data source [25].

Synthetic data constraints. We use three constraints on imputed queue lengths $\hat{T}_r$ for every coarse time interval T (50ms).

Measurement constraints are simple inversions of the known functions used for coarsening (i.e., the monitoring tools). Concretely, we require that the maximum value of the imputed queue length time series at every interval equals the value that LANZ reported m_max, and the instantaneous queue length m_len_t equals the value of the periodic sampling at t^{th}ms.

$$\max_{0 \le t < T} \hat{T}_r[t] = m\_max \tag{C1}$$

$$\forall t \in T_{samples}. \quad \hat{T}_r[t] = m\_len_t \tag{C2}$$

Operational constraints express the connection between switch operation and the counts of packets sent from SNMP measurements. If a queue is nonempty for NE ms, then at least NE packets have been dequeued, as schedulers are work-conserving. An empty queue can send a packet if one arrives; hence NE is a lower bound on packets sent count (m_out).

$$NE \le m\_out \tag{C3}$$

$$where \quad NE = \sum_{t=0}^{T-1} ite(\hat{T}_r[t] > 0, \quad 1, \quad 0)$$

Meta data constraints. We formulate four constraints for imputed link utilization $\hat{T}_r$ for every coarse time interval T (*i.e.,* 50ms). Measurement constraints come from the measurement of aggregated traffic rates m_sum.

$$\sum_{t=0}^{T-1} \hat{T}_r[t] = m\_sum \tag{C4}$$

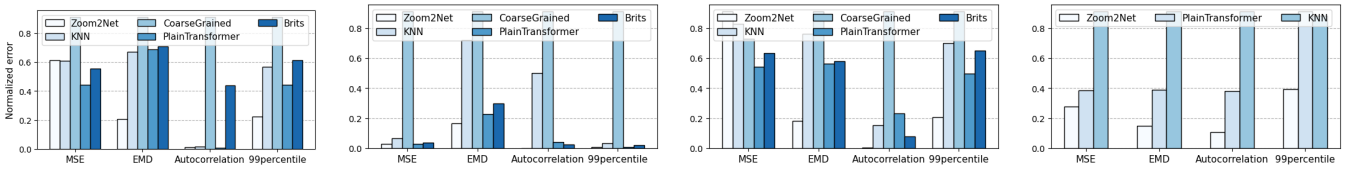
Operational constraints for imputed link utilization data articulate its relationships with congestion and retransmission. The imputed link utilization should be at least the number of bytes in both congestion ($congestion\_sum$) and retransmitted ($retransmit\_sum$) scenarios. In the presence of congestion during a 50ms interval, there should be at least one burst observed in the imputed link utilization.

$$\sum_{t=0}^{T-1} \hat{T}_r[t] \ge retransmit\_sum \tag{C5}$$

$$\sum_{t=0}^{T-1} \hat{T}_r[t] \ge congestion\_sum \tag{C6}$$

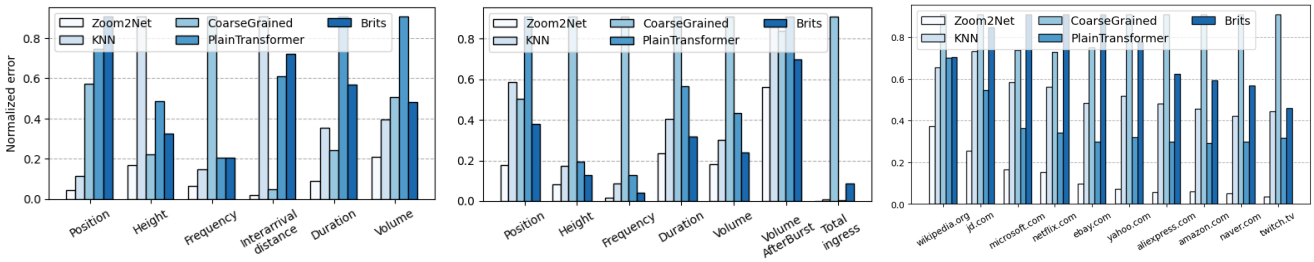
$$congestion\_sum > 0 \rightarrow \max_{0 \le t < T} \hat{T}_r[t] \ge \frac{1}{2} bandwidth \tag{C7}$$

Zoom2Net captures the structural pattern of the fine-grained time series better than baselines, although it results in higher MSE (Q2). Fig. 8a and Fig. 8b summarize the metrics for Zoom2Net accuracy on both datasets. Zoom2Net outperforms baselines in EMD, autocorrelation, and the 99th percentile by 33–53% in the synthetic dataset. Unsurprisingly, the plain transformer, which is trained with MSE, has the lowest error in MSE, as it generates overly smooth and, hence, low values for queue lengths. On the contrary, the incorporation of EMD in the loss function prompts Zoom2Net to de-emphasize minor shifts in burst positions, which hurts MSE but



(a) Case-1 Synthetic (b) Case-1 Meta (c) Case-2 MLab (d) Case-3 VPN

Figure 8: Zoom2Net outperforms baselines in EMD, Autocorrelation, and 99p accuracy, leading to superior performance in downstream tasks. The plain transformer achieves the lowest MSE by generating overly smooth results.



(a) Burst analysis for Case-1 synthetic (b) Burst analysis for Case-1 Meta (c) Page loading time for Case-2 MLab

Figure 9: Normalized error for downstream task performance. (a) and (b) show that Zoom2Net effectively captures burst behaviors in both synthetic and real-world datasets. (c) demonstrates that Zoom2Net precisely estimates webpage loading time for various websites by imputing accurate sending rates. Overall, Zoom2Net surpasses baselines by 38%.

encourages more accurate imputation of the burst shape. For Meta data, the three metric accuracies (EMD, autocorrelation, and the 99th percentile) of Zoom2Net imputed link utilization is better than baselines by a margin of 33%.

Zoom2Net effectively recovers bursts, outperforming other baselines in all downstream tasks (Q2). As shown in Fig. 9a, Zoom2Net achieves a significant improvement in burst properties ranging from 10% to 88% over all baselines in all tasks for synthetic data. Notably, Zoom2Net attains an average error of only 4% for burst position, even though Zoom2Net loss (including EMD) treats slightly shifted bursts as equivalent, demonstrating its effectiveness in capturing network dynamism. Across burst analysis for the real-world data in Fig. 9b, Zoom2Net exhibits an average performance superiority of 30%. There are cases where certain baselines perform comparably with Zoom2Net, such as plain transformer and Brits. This can be attributed to the characteristics of the dataset. The data is skewed towards smaller values with the infrequent occurrence of bursts lasting 1-2ms, leading to low errors in the 99th percentile, autocorrelation, burst height and frequency.

Zoom2Net enhances performance compared to directly using coarse-grained data (Q1). Zoom2Net achieves up to 5 times better performance in downstream tasks compared to using coarse-grained data directly. This highlights the critical role of Zoom2Net in learning from data to improve downstream tasks.

Zoom2Net leverages correlations learned in different settings. The synthetic testing dataset includes combinations of traffic patterns and congestion control algorithm settings that were not present in the training set. Remarkably, Zoom2Net performs even better on these unseen settings by an average of 30% compared to the scenarios in the training set. This underscores Zoom2Net's capacity to apply learned correlations effectively to diverse, previously unseen scenarios.

6.3 Case study 2: CDN PoP selection

Consider a Content Delivery Network (CDN) operator managing multiple Point-of-Presence (PoP) and serving different webpages to users. When determining the optimal PoP for serving diverse webpages to diverse users, the operator needs to estimate the time it takes for different amounts of data (corresponding to each webpage) to reach users from each PoP. Observe that because of congestion control and the heterogeneity of networks, predicting time to transfer is not trivial, i.e., it is not a linear connection. The operator can calculate the time required to send varying amounts of webpage data from each PoP to each user around the globe by observing the sending rate of the user-PoP pair over time. In practice, the operator cannot send varying amounts of data to users and record the time of reach. Instead, they have access to user-initiated network speed tests using Network Diagnostic Tools (NDT) at a coarser granularity.

MLab Datasets. For this demonstration, we leverage data from the M-Lab project's NDT measurements [26]. The NDT measurements capture TCPInfo and BBRInfo statistics from each snapshot of a data transfer that spans 250ms on average. These statistics include achieved throughput, minimum RTT, bytes sent, retransmitted, and more. M-Lab also provides packet traces recorded during NDT measurements. These traces provide transmission rates, forming the fine-grained time series at 10ms granularity. Our training set contains 5000 data points. **Imputation goal:** Zoom2Net leverages coarse-grained NDT measurements recorded at *250ms* intervals to impute fine-grained sending rates at a *10ms* granularity. We profile the number of bytes loaded by 10 Top Alexa websites, excluding failed ones. We calculate the cumulative distribution of sending rates to estimate the time required to load the websites.

Constraints. We formulate two measurement constraints related to the maximum sending rate and aggregated sending traffic volume, expressed similarly as ($C1$) and ($C4$).

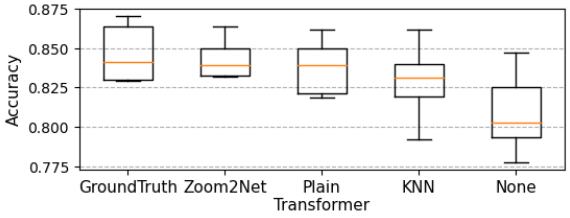


Figure 10: Zoom2Net improves accuracy and stability of VPN traffic classification by recovering features from imputed traces.

For operational constraints, the first one ensures that traffic volume is at least the number of bytes retransmitted, similar to ($C5$). Furthermore, when the NDT measurement period ($elapsed\_time$) is shorter than the RTT, the sending traffic should not exceed the product of the Maximum Segment Size (MSS) and congestion window size ($SndCwnd$). If the measurement period is shorter than the time spent waiting for the receiver window ($RwndLimited$), indicating the sender waiting and pauses sending, the traffic rate should remain at 0.

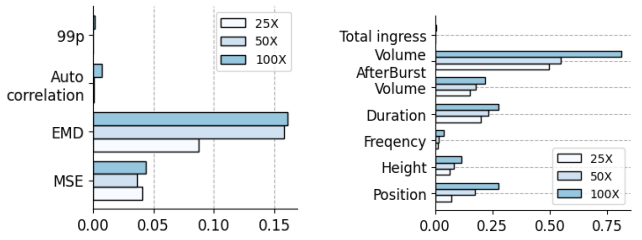
$$elapsed\_time \leq RTT \rightarrow \sum_{t=0}^{T-1} \hat{T}_r[t] \leq MSS \times SndCwnd \quad (C8)$$

$$elapsed\_time \leq RwndLimited \rightarrow \sum_{t=0}^{T-1} \hat{T}_r[t] = 0 \quad (C9)$$

Zoom2Net's fine-grained output provides an accurate estimate of page loading time to users around the globe (Q1, Q2). Fig. 9c illustrates the average error in page loading time estimation for 10 top Alexa websites, sorted by the number of bytes they load. The amount of data transmitted by these websites ranges from 300KB to 10MB. Zoom2Net shows an average improvement of 43% in accuracy in loading time estimation compared to other baselines. We observe that Zoom2Net's accuracy improves with an increasing number of bytes loaded. This is due to the normalization of smaller values to even smaller scales, where Zoom2Net is less effective at capturing minor variations. In Fig. 8c, Zoom2Net shows better performance by 44% on average across statistics metrics.

6.4 Case study 3: Encrypted traffic classification

Consider a network practitioner tasked with detecting encrypted VPN traffic based on flow-based time-related features, such as flow duration, maximum, minimum forward/backward inter-arrival time, and packet size [21]. The practitioner had access to a packet trace, which they used to extract these features and train their model. Then they discarded the trace due to its impractically large size for storage and privacy concerns. Some months later, the practitioner discovers new features useful for classification, but they cannot retrain because the original traces are no longer available for additional feature extraction. Zoom2Net offers a solution in this scenario. When the practitioner initially extracts features from the trace, they can use these features as coarse-grained input for Zoom2Net to impute the packet trace, aiming to recover details such as the arrival time and length of each packet. The practitioner can then keep the Zoom2Net model and discard the trace. Later, they can extract new features



(a) Normalized relative error of imputation accuracy

(b) Normalized relative error of downstream task accuracy

Figure 11: Zoom2Net does not exhibit substantial performance degradation under different zoom-in factors.

from the imputed trace and add to classification tasks. This provides the flexibility to extract new features without the overhead of storing large packet traces, enabling a more adaptive approach to feature engineering and classification.

VPN Datasets. In this case study, we begin with a real-world packet trace dataset [21]. From this data, we extract features such as maximum and minimum forward/backward inter-arrival time, and maximum packet length which serve as the coarse-grained input. We formulate the fine-grained data, specifically packet arrival time and packet length by parsing the trace. We use a training set of 3,300 data points. **Imputation goal:** Zoom2Net uses a *single* set of coarse-grained features extracted from packet traces to generate fine-grained packet trace information of averaged *20* packets. From the imputed packet trace information, we extract additional features flow duration and flow rates, and add them to the initial features for classification.

Constraints. We utilize measurement constraints specific to the minimum and maximum of forward/backward inter-arrival time, and maximum of packet length. They are formulated similar to ($C1$).

Zoom2Net can recover features by imputing packet traces, improving traffic classification accuracy (Q1, Q2). In Fig. 10, we report the classification accuracy of a multi-layer perceptron model that uses the union of ground-truth features calculated on the initial trace with additional features calculated on the imputed trace. The different bars illustrate the source of the additional features: ground-truth traces, KNN-imputed traces, plain transformer-imputed traces, and the case where no additional feature is added. Each scenario runs for 10 times. We observe that the classification accuracy using features extracted from Zoom2Net-imputed traces is comparable to that extracted from the ground truth and higher than scenarios without additional features, demonstrating the effectiveness of Zoom2Net in capturing the correlations and characteristics of traces. Notably, the classifier trained with Zoom2Net-imputed features is more stable. The minimum and the first quartile of Zoom2Net's accuracy are the highest among baselines. For statistical metrics in Fig. 8d, Zoom2Net demonstrates a notable average performance improvement of 53% over other methods. The given dataset lacks periodic samples, restricting the applicability of pure coarse-grained data and Brits so we did not compare with them.

6.5 Zoom-in factor analysis

In this section, we evaluate the performance of Zoom2Net under different zoom-in factors, defined as the ratio of coarse granularity

Window Size (second)	Zoom-in Factor	Transformer	Transformer +CEM
0.25	50	0.00054	0.145
0.5	50	0.00056	0.215
1	50	0.00094	0.394
1	25	0.00070	0.532
1	100	0.00052	0.285

Table 1: Zoom2Net running time in seconds in case study §6.2

and fine granularity. Using the Meta dataset from §6.2, we evaluate three zoom-in factors: 25, 50, and 100. We calculate their relative error compared to the ground truth and normalize the error using the same normalization factor as in §6.2 to facilitate comparisons with other baseline methods. When we increase the zoom-in factor from 50 to 100, Fig. 11 shows that imputation accuracy increases by 0.4% and downstream tasks accuracy increases by 7%. Remarkably, the imputation accuracy with a factor of 100 outperforms baselines with a factor of 50 by an average of 32%. The overall performance remains superior to baselines by 23% on average. These results underscore Zoom2Net's capability to effectively impute super coarse-grained data to fine-grained measurements.

7 Discussion

In this section, we discuss practical considerations regarding Zoom2Net. First, we report the running time of Zoom2Net, which is critical to determine its suitability for real-time applications, even though real-time operation is not within the scope of the paper. We have found that the CEM (Consistency Enforcement Module) dominates the running time. Next, we present a technique to quantify and reveal the model's confidence, a feature that can be very useful for operators.

Timing analysis The running time of Zoom2Net is dominated by the CEM. Indeed, running the transformer inference alone takes less than 1ms, while the entire system takes less than 0.55s in all use cases we evaluate. CEM's running time varies significantly across different use cases as it depends on the corresponding constraints. For example, case-1 synthetic (§6.4) takes 0.15s less compared to case-1 Meta (§6.2) with the same zoom-in factor and window size.

Time-window size and zoom-in factor also affect the running time, although not as much as CEM. We observe that the overall system running time increases with larger window sizes and smaller zoom-in factors, due to a higher number of intervals that need correction. We run the case study in §6.2 with Meta's data using different combinations of. Table 1 summarizes the running time of Zoom2Net in seconds with and without CEM and with different combinations of window sizes and smaller zoom-in factors.

Model Confidence In deploying Zoom2Net, it is crucial for operators to gauge the model's confidence in its predictions to determine its reliability. A lower confidence level might suggest the need for human analysis rather than solely relying on the automated output. To facilitate this, we incorporate Monte Carlo dropout as a method for estimating uncertainty, as described in [23]. Originally used during training to prevent overfitting, dropout also serves as an approximation to a probabilistic deep Gaussian process when applied during inference. By enabling dropout during inference and executing multiple stochastic forward passes through the model, we can assess variability in the predictions. A high variance among these

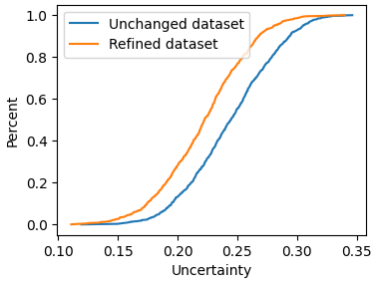


Figure 12: Training dataset refinement helps reduce model uncertainty.

passes indicates a higher uncertainty. For each input, we conduct 100 forward passes and use the standard deviation of these outputs as the uncertainty score. One typical source of uncertainty is the inherent noise in the training data [30]. In our case, coarse-grained time series collisions (§3.2) can lead to model uncertainty. In Fig 12, we show the effect of using the refined training dataset on the uncertainty of the model described in case study §6.2. The refinement module in §4.4 allows the model to learn essential correlations and improve the certainty of its output.

Generalization Beyond the three use cases discussed in Section 6, we anticipate that our approach can be applied to a wide variety of real-world scenarios. Specifically, Zoom2Net is likely to be effective across any set of correlated network time series, provided there is access to a small, fine-grained dataset for these series and the specific functions of the coarsening operators (i.e., the monitoring tools) are known. The performance of Zoom2Net can be significantly enhanced by mathematically articulating some of the correlations among the time series. Training Zoom2Net periodically would be beneficial to mitigate the effects of distribution shifts [29].

8 Conclusion

This paper presents a new paradigm for network telemetry. Instead of improving hardware or collection algorithms, we advocate for post-collection software telemetry imputation. We present Zoom2Net, an ML-based system that analyzes multiple correlated coarse-grained time series to impute their fine-grained counterparts. What sets Zoom2Net apart is its incorporation of domain knowledge through operational and measurement constraints. We explored several use cases for Zoom2Net using synthetic and public datasets and demonstrated Zoom2Net's capability to accurately impute diverse types of telemetry data. The results highlight the effectiveness of Zoom2Net in facilitating reliable downstream tasks.

Acknowledgements

We thank our shepherd Prof. Athina Markopoulou and the anonymous SIGCOMM reviewers for their constructive feedback and suggestions. This work was supported by the National Science Foundation (NSF) through Grants CNS-2319442 and CNS-2312539, a Google Research Scholar Award, and a Princeton Innovation Fund Award.

References

[1] 2016. Arista LANZ Overview. https://www.arista.com/assets/data/pdf/Whitepapers/Arista_LANZ_Overview_TechBulletin_0213.pdf. (2016).

[2] 2024. Gurobi solver. (2024). https://www.gurobi.com/.

[3] 2024. ns3-datacenter. https://github.com/inet-tub/ns3-datacenter. (2024).

[4] 2024. NS3 Network Simulator. https://www.nsnam.org/. (2024).

[5] 2024. Trident2 / BCM56850 Series, High-Capacity StrataXGS® Trident II Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56850-series. (2024).

[6] Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. 2022. ABM: Active buffer management in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 36–52.

[7] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. 2024. Reverie: Low Pass Filter-Based Switch Buffer Sharing for Datacenters with RDMA and TCP Traffic. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), Santa Clara, CA*.

[8] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. 2021. Performance-Driven Internet Path Selection. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 41–53.

[9] Yuval Bahat and Tomer Michaeli. 2020. Explorable Super Resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[10] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic In-band Network Telemetry *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3387514.3405894

[11] Henry Birge-Lee, Sophia Yoo, Benjamin Herber, Jennifer Rexford, and Maria Apostolaki. 2024. TANGO: Secure Collaborative Route Control across the Public Internet.. In *NSDI*.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[13] Hans Buehler, Blanka Horvath, Terry Lyons, Imanol Perez Arribas, and Ben Wood. 2020. A data-driven market simulator for small data environments. *arXiv preprint arXiv:2006.14498* (2020).

[14] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. 2018. BRITS: Bidirectional Recurrent Imputation for Time Series. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/734e6bfcd358e25ac1db0a4241b95651-Paper.pdf

[15] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. 2019. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 15–29. https://doi.org/10.1145/3359989.3365408

[16] Zhuo Cheng, Maria Apostolaki, Zaoxing Liu, and Vyas Sekar. 2024. TRUSTSKETCH: Trustworthy Sketch-based Telemetry on Cloud Hosts. In *The Network and Distributed System Security Symposium (NDSS)*.

[17] Abhijit K Choudhury and Ellen L Hahne. 1998. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions On Networking* 6, 2 (1998), 130–140.

[18] Alexander Dietmüller, Siddhant Ray, Romain Jacob, and Laurent Vanbever. 2022. A New Hope for Network Model Generalization. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 152–159.

[19] Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. 2022. Neural Networks with Physics-Informed Architectures and Constraints for Dynamical Systems Modeling. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference (Proceedings of Machine Learning Research)*, Vol. 168. PMLR, 263–277. https://proceedings.mlr.press/v168/djeumou22a.html

[20] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence* 38, 2 (2015), 295–307.

[21] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 407–414.

[22] Mark Fedor, Martin Lee Schoffstall, James R. Davin, and Dr. Jeff D. Case. 1990. Simple Network Management Protocol (SNMP). RFC 1157. (May 1990). https://doi.org/10.17487/RFC1157

[23] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 48. PMLR, New York, New York, USA, 1050–1059. https://proceedings.mlr.press/v48/gal16.html

[24] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 549–564.

[25] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. 2022. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*. Association for Computing Machinery, New York, NY, USA, 567–580. https://doi.org/10.1145/3517745.3561430

[26] Phillipa Gill, Christophe Diot, Lai Yi Ohlsen, Matt Mathis, and Stephen Soltesz. 2022. M-Lab: User initiated Internet data for the research community. *ACM SIGCOMM Computer Communication Review* 52, 1, 34–37.

[27] Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2023. Towards Integrating Formal Methods into ML-Based Systems for Networking. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 48–55.

[28] Zied Ben Houidi, Raphael Azorin, Massimo Gallo, Alessandro Finamore, and Dario Rossi. 2022. Towards a Systematic Multi-Modal Representation Learning for Network Data. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 181–187.

[29] Kevin Hsieh, Mike Wong*, Santiago Segarra, Sathiya Kumaran Mani, Trevor Eberl, Anatoliy Panasyuk, Ravi Netravali, Ranveer Chandra, and Srikanth Kandula. 2024. NetVigil: Robust and Low-Cost Anomaly Detection for East-West Data Center Security. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[30] Willem Hüllermeier, Eyke andWaegeman. 2021. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning* 110 (March 2021), 457–506. https://doi.org/10.1007/s10994-021-05946-3

[31] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2023. Generative, High-Fidelity Network Traces. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 131–138. https://doi.org/10.1145/3626111.3628196

[32] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. 2021. Physics-informed machine learning. *Nature Reviews Physics* 3, 6 (2021), 422–440.

[33] Aleksandar Kuzmanovic and Edward W Knightly. 2003. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the ACM SIGCOMM 2003 conference*. 75–86.

[34] Jonatan Langlet, Ran Ben Basat, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. 2023. Direct Telemetry Access. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3603269.3604827

[35] Franck Le, Mudhakar Srivatsa, Raghu Ganti, and Vyas Sekar. 2022. Rethinking Data-Driven Networking with Foundation Models: Challenges and Opportunities. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 188–197.

[36] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[37] Yiran Lei, Liangcheng Yu, Vincent Liu, and Mingwei Xu. 2022. PrintQueue: performance diagnosis via queue measurement in the data plane. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 516–529. https://doi.org/10.1145/3544216.3544257

[38] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. {FlowRadar}: A Better {NetFlow} for Data Centers. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*. 311–324.

[39] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. 2020. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*. 464–483.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, B. Michel, V.and Thirion, O. Grisel, M. Blondel, R. Prettenhofer, P.and Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[41] Mimi Qian, Lin Cui, Fung Po Tso, Yuhui Deng, and Weijia Jia. 2023. OffsetINT: Achieving High Accuracy and Low Bandwidth for In-Band Network Telemetry. *IEEE Transactions on Services Computing* (2023), 1–12. https://doi.org/10.1109/TSC.2023.3323697

[42] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. 2014. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 407–418.

[43] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. 2022. Continuous In-Network Round-Trip Time Monitoring. In *Proceedings of the ACM SIGCOMM 2022 Conference*. New York, NY, USA, 473–485. https://doi.org/10.1145/3544216.3544222

[44] Renata Teixeira, Aman Shaikh, Timothy G Griffin, and Jennifer Rexford. 2008. Impact of hot-potato routing changes in IP networks. *IEEE/ACM Transactions On Networking* 16, 6 (2008), 1295–1307.

[45] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. 2018. Stroboscope: Declarative network monitoring on a budget. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 467–482.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[47] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. 2021. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1905–1914.

[48] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. 2018. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*. 0–0.

[49] Jackson Woodruff, Andrew W Moore, and Noa Zilberman. 2020. Measuring Burstiness in Data Center Applications. In *Proceedings of the 2019 Workshop on Buffer Sizing*. 6.

[50] Ming Wu, Qianmu Li, Fei Yang, Jing Zhang, Victor S Sheng, and Jun Hou. 2023. Learning from biased crowdsourced labeling with deep clustering. *Expert Systems with Applications* 211 (2023), 118608.

[51] Ming Wu, Qianmu Li, Jing Zhang, and Jun Hou. 2022. Label Aggregation with Clustering for Biased Crowdsourced Labeling. In *Proceedings of the 2022 14th International Conference on Machine Learning and Computing (ICMLC '22)*. Association for Computing Machinery, New York, NY, USA, 165–169. https://doi.org/10.1145/3529836.3529861

[52] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. 2022. Practical GAN-based synthetic IP header trace generation using NetShare. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3544216.3544251

[53] Kai Zhang, Luc Van Gool, and Radu Timofte. 2020. Deep unfolding network for image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3217–3226.

[54] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2472–2481.