Towards Integrating Formal Methods into ML-Based Systems for Networking

Fengchen Gong
Princeton University

Divya Raghunathan
Princeton University

Aarti Gupta
Princeton University

Maria Apostolaki
Princeton University

Abstract

Owing to its adaptability and scalability, Machine Learning (ML) has gained significant momentum in the networking community. Yet, ML models can still produce outputs that contradict knowledge, *i.e.,* established networking rules and principles. On the other hand, Formal Methods (FM) use rigorous mathematical reasoning based on knowledge, but suffer from the lack of scalability. To capitalize on the complementary strengths of both approaches, we advocate for the integration of knowledge-based FM into ML-based systems for networking problems. Through a case study, we demonstrate the benefits and limitations of using ML models or FM alone. We find that incorporating FM in the training and inference of an ML model yields not only more reliable results but also better performance in various downstream tasks. We hope that our paper inspires a tighter integration of FM-based and ML-based approaches in networking, facilitating the development of more robust and dependable systems.

CCS Concepts

• **Networks → Network monitoring**.

Keywords

Telemetry, Imputation, Formal Methods, Transformer

ACM Reference Format:
Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2023. Towards Integrating Formal Methods into ML-Based Systems for Networking . In *The 22nd ACM Workshop on Hot Topics in Networks (HotNets '23), November 28–29, 2023, Cambridge, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3626111.3628188

1 Introduction

Advances in Machine Learning (ML) have been disruptive in multiple domains, spanning natural language processing, computer vision, and recommendation systems. Networking has also become a playground for ML, thanks to its multiple intricate and largely unresolved problems, complex patterns, and abundance of data. Consequently, ML-based algorithms have been proposed for handling a wide range of long-standing networking challenges, including congestion control [19, 31], anomaly detection [4, 49], synthetic data generation [41], and performance forecasting [43].

While scalable, adaptable, and often more efficient than traditional solutions, ML-based algorithms lack correctness guarantees and are susceptible to overfitting patterns or settings. Thus, ML models could produce results that contradict domain knowledge, meaning that they are inferior to primitive algorithms, defy physical laws, or contradict common sense [1, 21]. For instance, in predicting delay, certain outcomes are implausible, such as those exceeding the speed of light. Similarly, in synthetic traffic-trace generation, the traffic rate originating from a port could not surpass its capacity.

It is evident that *knowledge* represented as mathematical equations, physical laws, probabilistic relationships, or logic rules should be somehow incorporated into solutions. Formal Methods (FM) is, of course, the time-honored way of leveraging knowledge in generating correct and sound results. As a result, FM has also seen significant success in networking problems such as congestion control [9], configuration synthesis [20, 50], traffic engineering [27], and finding bugs [32, 34, 37]. Unfortunately, FM-based networking solutions have difficulty in scaling and in learning patterns.

To get the best of both worlds, this paper advocates for the integration of FM into ML-based networking to enhance its dependability. Doing so could foster more reliable models with soundness guarantees, which could be trained with less data, and generalize better. While FM has been proposed as a mechanism to enhance confidence in trained models based on Deep Reinforcement Learning (DRL) [21, 57] and domain knowledge drives the *selection* of the model and/or the features in ML-based networking [57, 58], we take one step further and argue for *knowledge-augmented training and inference*. In fact, knowledge-augmented models are already proposed in physics and biology with Physics- [35] (or Biology- [38]) Informed Neural Networks.

To investigate the potential of the integration of FM into ML-based networking, we focus on a problem of network telemetry, but our methodology can translate to other networking problems. In particular, we impute fine-grained queue-length time series in switches by jointly analyzing their coarse-grained counterparts, together with other time series such as packet and drop counts. This is a suitable use case. First, fine-grained queue monitoring is crucial for multiple downstream tasks such as anomaly detection, provisioning, and root-cause analysis. Second, fine-grained queue monitoring is challenging due

HotNets '23, November 28–29, 2023, Cambridge, MA, USA

ACM ISBN 979-8-4007-0415-4/23/11.
https://doi.org/10.1145/3626111.3628188

to hardware limitations and scale [13, 24, 60]. Finally, queue lengths are affected by traffic patterns and follow well-studied principles.

At first glance, this problem could be solved by either ML or FM. On the one hand, imputing time series can be seen as an analogy to image superresolution (*i.e.,* turning a low-resolution image to a high-resolution one), which is often solved using generative models in ML [18, 40, 53, 54, 59, 62]. However, we find that using ML alone yields results that are often evidently inaccurate, *i.e.,* inconsistent with measurements or against known principles, especially for uncommon incidents. On the other hand, given a set of measurements (coarse-grained time series), and a set of constraints connecting them to their fine-grained counterparts and to each other, an FM model could, in principle, compute a fine-grained time series. Unfortunately, we find that such a solution is hard to scale, due to the large search space.

While seemingly straightforward, combining FM and ML for queue length imputation poses multiple challenges which also generalize to other networking problems. First, there is no standard way to incorporate domain knowledge into ML models. Second, incorporating knowledge can significantly increase the complexity of the learning process, which might cause scalability issues for complex models. Third, designed to learn from data, ML models cannot easily ingest traditional knowledge such as rules and relationships.

To address these challenges, we start from a pure ML-based approach and strategically incorporate some of the constraints of our FM model in its training and inference. The ML-based approach (a transformer) ingests multiple sampled (coarse-grained) switch-level time series and outputs fine-grained queue lengths. Specifically, we first select constraints that can be directly evaluated on the transformer output. Doing so maintains system scalability because the system does not need to reason about detailed (per-packet) scenarios. Next, we transform those constraints to a differentiable form such that they can be incorporated into the loss function of the transformer. Finally, we enforce the constraints that the transformer failed to satisfy by correcting its output post-imputation. We show that combining ML with FM effectively increases queue-length monitoring granularity by 50x (from 50ms to 1ms) and yields 11-96% better results compared to ML alone.

We posit that knowledge-augmented ML-based models for networking instigate a vibrant research trajectory. This trajectory includes questions such as: Which networking problems require both ML and FM to be solved? How do we represent decades of accumulative knowledge on network calculus, network tomography, and optimizations in an ML-friendly way? How to use knowledge to fight the scarcity or bias of datasets? How can we verify that an ML system has indeed learned networking principles?

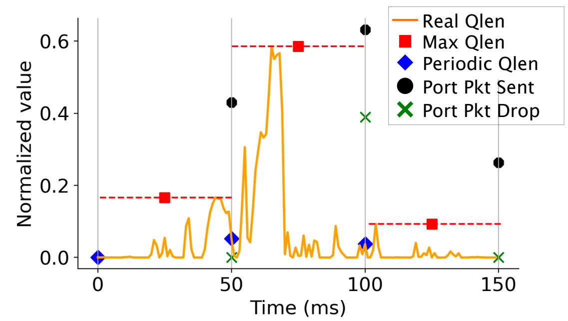


Figure 1: Sampling the queue length hides significant insights. The various coarse-grained time series are correlated e.g., drop increases with queue length.

2 Case study

To demonstrate the potential of the integration of ML-based networking with FM, we consider the task of fine-grained monitoring of queue lengths in network switches. Queue monitoring is a particularly useful but challenging task. Queue lengths affect latency guarantees [63] and expensive on-chip buffer provisioning [56]. However, queue lengths can change at the microsecond granularity making queueing the most unpredictable part of a packet's journey [44] and monitoring them particularly hard [13]. Instead of investigating hardware upgrades and/or telemetry operators, we work on a purely software approach. Concretely, we seek to use existing coarse-grained time series of various signals to impute fine-grained queue lengths.

2.1 Example Scenario

Consider an operator of a large datacenter 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, ideally, the operator needs to have fine-grained (say microsecond level) measurements of queue length of each switch over time. Indeed, longitudinal analyses of fine-grained queue length measurements will give the operator an idea of the common burst sizes and frequencies to inform the trade-off between accommodating bursts and reducing switch cost [55].

Let us assume, without loss of generality, that the datacenter has output-queued switches with N ports, each with two queues and a shared buffer across all queues, as shown in Fig. 2. In an ideal case, the operator would collect fine-grained time series of the length for each queue, $Q = \{Q_1, \ldots Q_{2N}\}$ and use that for the downstream tasks. In practice, the operator has access to monitoring only at a much coarser granularity, say, every 50ms. Consider, for instance, that the operator can use *(i)* periodic sampling of each queue, which provides instantaneous queue lengths; *(ii)* LANZ [8], which provides the per-queue maximum length within each interval[1], but does not specify exactly when the maximum occurred; and *(iii)* SNMP [22],

[1]In practice, LANZ will only monitor a queue after its length exceeds a configurable threshold, but for simplicity, we assume that the operator configured this low enough s.t. LANZ reports some value for each interval unless the queue is constantly zero.

which provides per-port counts of packets sent, and dropped every interval. As an illustration, Fig. 1 shows a queue length time series at fine granularity (continuous line) and the measurements available to the operator (dots). Arguably, sampling hides critical details.

Insights: While each routinely collected coarse-grained time series (*i.e.,* queue lengths, packet counts, and drops) alone hides important incidents, we posit that we can discover more about the network by analyzing them together. In fact, all these series follow the operating principles of a switch and are affected by common traffic, thus are correlated. First, as the buffer is shared across multiple queues, their lengths are correlated: a longer queue prevents other queues from growing by taking up space in the buffer [2, 3, 6, 14]. Second, loss rate and queue lengths are correlated: loss only occurs when queues are longer than a threshold. Finally, queue lengths are correlated with the incoming packet rate: a queue only forms in a many-to-one (fan-in) scenario, *i.e.,* when the service rate is lower than the incoming rate. As shown in Fig. 1, an increase in the queue length is accompanied by an increase in the coarse-grained packets sent and dropped in the same interval.

These insights beg the question: *Can we use the available coarse-grained monitoring to impute fine-grained queue lengths?* One strategy is to train an ML model (*e.g.,* a transformer) to predict fine-grained queue lengths given the coarse-grained time series (§2.2). Another strategy is to model a switch using Satisfiability Modulo Theory (SMT) [10] constraints and use an SMT solver [15] to find fine-grained queue lengths consistent with the measurements (§2.3).

2.2 Telemetry Imputation with ML

Next, we explain why using ML is a natural choice and what challenges we encountered in the process.

Why ML is a good idea: ML techniques are good at handling complex and non-linear relationships between variables by learning directly from data. This allows us to leverage correlations that are too complex to model or for which we do not have perfect knowledge. For example, for predicting queue length, one would need to model congestion control, dynamic buffer sharing, scheduling, and even traffic patterns and demands. Moreover, ML is scalable: models can be parallelized in the training and inference phases.

We find transformers to be particularly suitable models for telemetry imputation. A transformer is a sequence-to-sequence model that is able to learn correlations over a long sequence in parallel based on the attention mechanism [52]. Its flexibility and efficiency have already made it popular in the networking context [16, 29, 39, 58].

Why ML is *not* a good idea: The most significant downside of using ML for telemetry imputation is that the output evidently lacks correctness. As an illustration, Fig. 4b shows the output of a transformer we trained to impute queue lengths from coarse-grained time series (§4 provides details for the model and data generation). We observe that the imputed time series (blue line) is not consistent with the measurements. For example, the transformer did not impute a queue length that is as high as the (known) max queue length (red dot) of the interval between 50-100ms, although it was part of the transformer's input. It may seem surprising that the model does not "realize" the connection between the provided max 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, we found that our model also violated 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.

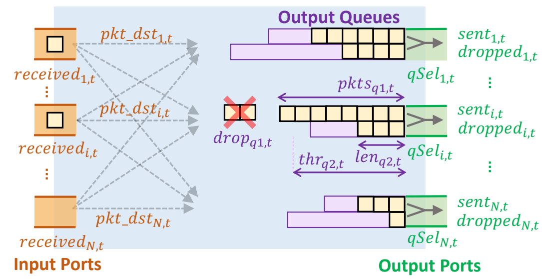


Figure 2: Schematic diagram of an output-queued switch. The annotations refer to variables in our FM model (§2.3).

2.3 Telemetry Imputation with FM

The inability of our ML model to produce a fine-grained time series consistent with our knowledge, *i.e.,* the coarse-grained measurements and domain-specific rules, motivates the use of FM for telemetry imputation.

Why formal methods is a good idea: FM allows us to express our knowledge about how the switch operates and use automated reasoning to find a scenario that fits the coarse-grained observations. Importantly, FM provides a *guarantee* that a result is plausible, *i.e.,* could have occurred in a switch given the observed measurements and domain-specific constraints.

Inspired by prior work [7] on using FM to analyze performance in switches, we use Satisfiability Modulo Theories (SMT) constraints to model switch behavior at the level of a single packet. We divide time into discrete time steps, where a time step is the time taken to transmit or receive a packet. We then use Z3 [15] to find a solution to the system of constraints, which corresponds to a time series of queue lengths.

At a high level, we model two types of constraints: *(i)* operational, and *(ii)* measurement. The former describes the way a switch works. The latter incorporates the measurements, effectively demanding that the result of utilizing monitoring tools (*e.g.,* max queue length, packet counts) on the fine-grained time series would result in the coarse-grained measurements. Due to space limitations, we have omitted the formal equations.

Operational constraints. Every packet that arrives at the switch is mapped to some output queue, and a packet is dequeued if the scheduler selects some queue. If the queue was unbounded, the number of packets in the queue at time step t,

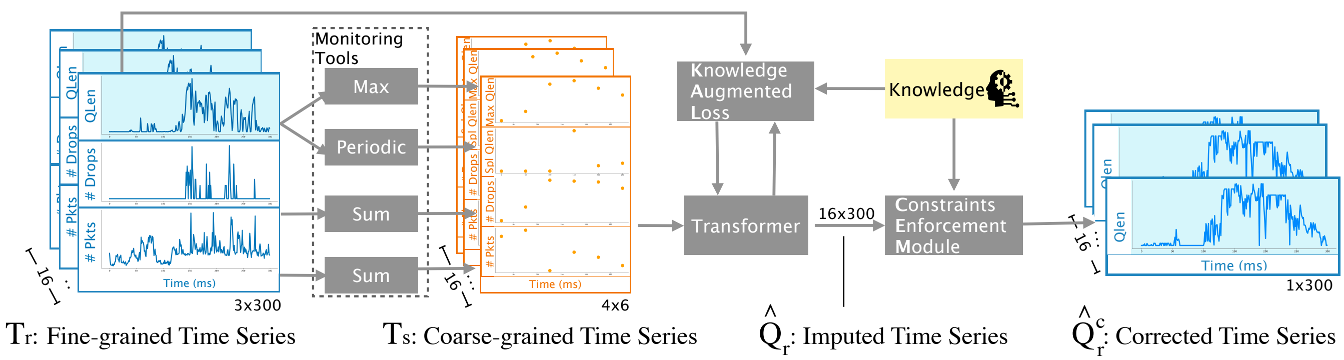


Figure 3: Queue lengths (highlighted) are sampled together with other time series by generic monitoring tools and input in the transformer which outputs fine-grained queue length time series. Knowledge (in the form of constraints) is used in the loss function during training (KAL) and on top of its output during inference (CEM).

$pkts^{\infty}_{q,t}$, is the sum of the queue length at $t-1$ and the number of packets received at t. In practice, the queue is bounded, thus if $pkts^{\infty}_{q,t}$ exceeds a dynamically calculated threshold ($thr_{q,t}$), the excess packets are dropped; the remaining packets are enqueued ($pkts_{q,t}$). The queue length at t is $pkts_{q,t}$ minus 1 if a packet is dequeued. Additional constraints over the selected queue and dynamic threshold model the scheduling and buffer management algorithms, respectively.

Measurement constraints. The number of packets received, sent, and dropped at each port must equal the counts reported by SNMP. The maximum queue length during the monitoring interval must equal the maximum as reported by LANZ (m_max_q). If the queue length is sampled at time steps $T_{samples}$, the imputed queue length must match the sampled queue length ($m\_len_{q,t}$).

Why formal methods alone is *not* a good idea. Although our FM approach guarantees the imputed result is plausible, its scalability is limited. Z3 successfully generated imputed queue lengths for simple scenarios in a few minutes, but could not handle more realistic scenarios in even 24 hours. Due to the high port bandwidth, there are ≈ 90 time steps in 1ms, leading to a large search space of traffic scenarios which makes the problem intractable for the solver. For instance, if two packets are enqueued sequentially on an empty queue, different inter-arrival gaps are considered, though they have the same effect on the queue length.[2]

3 Combining ML and FM

Driven by our previous observations demonstrating the complementary nature of FM and ML, we design a synergistic approach shown end-to-end in Fig. 3. The raw (fine-grained) time series T_r are sampled to T_s (*e.g.,* by generic monitoring tools) and fed to a transformer which is trained using a Knowledge-Augmented Loss (KAL) function. During inference, a Constraint Enforcement Module (CEM) corrects the transformer's output ($\hat{Q}_r$) by minimally changing it until it satisfies certain constraints producing $\hat{Q}^c_r$. Note that an operator would only have T_s available to infer $\hat{Q}^c_r$. For training, she can use a simulation or a short real trace to generate T_r.

Exact modeling as defined by the operational and measurement constraints is too expensive, as we discussed in §2.3. We navigate the trade-off between accuracy and scalability by reducing our system of constraints to three that we can directly test against the transformer-imputed queue lengths.

The first two constraints are related to measurements, particularly max and periodic sampling of queues:

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

$$\forall t \in T_{samples}. \quad \hat{Q}_r[q][t] = m\_len_{q,t} \tag{C2}$$

where $\hat{Q}_r[q][t]$ denotes the length of queue q at the t^{th} ms.

Finally, we observe that if some queue in port i is nonempty for NE_i time steps, then NE_i packets will be dequeued, as schedulers are typically work-conserving. An empty queue can send a packet if one arrives; hence NE_i is a lower bound on the number of packets sent, m_out_i:

$$NE_i \le m\_out_i \tag{C3}$$

$$\text{where } NE_i = \sum_{t=0}^{T-1} ite\Big(\bigvee_{q\in Queues_i} \hat{Q}_r[q][t] > 0, \quad 1, \quad 0\Big)$$

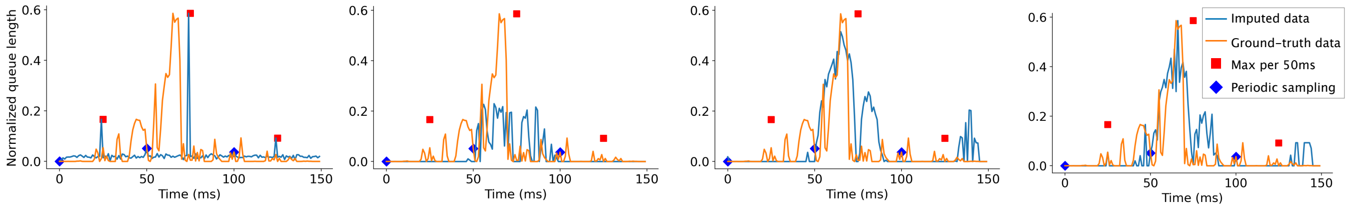
$ite(c,v_1,v_2)$ means if c is true, then return v_1, else return v_2.

The resulting set defines an *over-approximation* of the switch behavior: every plausible imputed result satisfies these constraints, but not every imputed result consistent with them is plausible. We integrate these constraints on the transformer's training and inference. For the training, we modify the loss function, effectively teaching the transformer to obey domain knowledge (§3.1). For the inference, we correct the transformer's output to be consistent with the constraints (§3.2).

3.1 Knowledge Augmented Loss (KAL)

Our transformer (§2.2) is trained with EMD (Earth Mover's Distance) [47] which allows it to learn queue-length distributions but not to satisfy known constraints. Thus, we augment the loss function to incorporate certain constraints. Besides

[2]This example scenario is on-purpose simplified, there are other cases that cannot be easily distinguished or pruned.



(a) IterativeImputer [48] **(b) Transformer-only** **(c) Transformer + KAL** **(d) Transformer + KAL + CEM**

Figure 4: Visualizations of imputing the same queue-length incident (described in §4) with various methods: (a) simply connects maximum and periodic queue-length samples; (b) catches trends but outputs results inconsistent with maximum and periodic samples; (c) approaches consistency (*e.g.,* queue length approaches known max) (d) consistent and accurate results.

$\text{EMD}(T_r, \hat{Q}_r)$, the transformer aims at satisfying both the equality constraints (C1, C2), and the inequality constraint (C3), presented by differentiable functions $\Phi(T_s, \hat{Q}_r)$ and $\Psi(T_s, \hat{Q}_r)$, respectively. To turn (C3) (which is non-differentiable as it involves an ite) into the differential Ψ, we *(i)* apply a Tanh function to each scaled queue length, resulting in 1 when the length is greater than 0, and 0 otherwise; and *(ii)* sum the results across all queues in the port to model the disjunction. The loss function is thus turned into a constrained optimization problem:

$$\min \text{EMD}(T_r, \hat{Q}_r) \text{ s.t. } \Phi(T_s, \hat{Q}_r) = 0,\ \Psi(T_s, \hat{Q}_r) \leq 0$$

To have the model learn to satisfy the constraints, we adopt the augmented Lagrangian method, inspired by [17]. This involves introducing penalty terms into the objective function to represent constraint violations. These penalty terms are weighted by Lagrange multipliers λ_i^{eq} for equality constraint Φ evaluated at each training example $(T_{s_i}, \hat{Q_{r_i}})$, and similarly λ_i^{ineq} for inequality constraint Ψ. At each training step, each Lagrange multiplier is updated by multiplying the violations of the corresponding output data by a parameter μ. The importance of a violation in the loss function increases as its magnitude becomes higher, requiring more effective minimization. Finally, the loss function is given by

$$\begin{aligned}\mathcal{L} = &\text{EMD}(T_r, \hat{Q}_r) + \sum_{i=1}^{N} \mu \Phi(T_{s_i}, \hat{Q_{r_i}})^2 \\ &+ \sum_{i=1}^{N} \lambda_i^{eq} \Phi(T_{s_i}, \hat{Q_{r_i}}) + \sum_{i=1}^{N} \lambda_i^{ineq} \Psi(T_{s_i}, \hat{Q_{r_i}}) \\ &+ \sum_{i=1}^{N} \mu [\lambda_i^{ineq} > 0 \vee \Psi > 0] \Psi(T_{s_i}, \hat{Q_{r_i}})^2\end{aligned}$$

where N is the training dataset size. Training with these penalty terms enables the model to learn the consistency between the input and output, enforced by the constraints. As shown in Fig. 4c, the imputed queue length of a transformer trained with KAL approaches the maximum value much more (compared to the pure transformer in Fig. 4b).

3.2 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 using the SMT solver Z3 (*i.e.,* forces it to satisfy the constraints we outlined before C1, C2, C3) while changing it as little as possible. We use variables $\hat{Q}_r^c[q][t]$ to denote the corrected queue lengths at each time step. To ensure that the corrected results remain 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 queue length is sampled.

$$\min \sum_{t=0,\ t \notin T_{samples}}^{T-1} |\hat{Q}_r^c[q][t] - \hat{Q}_r[q][t]|$$

Fig. 4d showcases the output generated using both KAL and CEM. In this case, we observe that the imputed values precisely follow the maximum values and periodic samples.

Observe that CEM does not significantly deteriorate the scalability of the system. First, selected constraints do not require the solver to calculate the state for every time step as in §2.3. Second, the transformer output has already satisfied some of the constraints, thanks to KAL.

4 Preliminary evaluation

We compare various imputation methods in terms of consistency and performance when their output is used for various downstream tasks. We find that a transformer augmented with CEM and KAL yields the best results while being scalable.

Imputation methods: We use four methods: a non-ML one and three transformer-based. First, we use the IterativeImputer [48], a statistical method that retains the periodic samples, models the feature with missing values as a linear function of other features iteratively. To feed IterativeImputer with the maximum queue length, we place the max at the midpoint of each interval. Second, we use a transformer encoder that encodes the set of coarse-grained time series and a linear layer as decoder to generate the fine-grained time series. We use EMD as our loss function as opposed to MSE because it improves the accuracy of the model in locating bursts. While MSE is more commonly used, we found that it encourages the model to find averages of plausible solutions that are overly smooth and is disadvantageous for bursts. Finally, we augment the transformer with KAL and with CEM as we describe in §3.

Error Metric	IterImputer	Transformer	Transformer +KAL	Transformer +KAL+CEM
a. Max Constraint	0.49	0.88	0.93	0
b. Periodic Constraint	0.078	0.16	0.15	0
c. Sent pkts count Constraint	0.43	0.52	0.025	0
d. Burst Detection	0.32	0.18	0.17	0.16
e. Burst Height	0.98	0.85	0.76	0.69
f. Burst Frequency	0.94	0.28	0.24	0.29
g. Burst Interarrival Time	6.33	2.83	0.13	0.1
h. Empty Queue Frequency	0.95	0.26	0.19	0.18
i. Avg count of concurrent bursts	0.53	0.12	0.09	0.08

Table 1: Downstream tasks' performance is significantly better when the transformer is augmented with KAL and CEM.

Data Generation: To generate realistic queue lengths we use the ns-3 simulator [45] for the scenario described in [2]. The generated traffic follows the websearch and incast traffic patterns. Each port is mapped to two queues with different classes. We collect fine-grained time series (ground-truth) corresponding to queue lengths, per-port packet and drop counts every 1ms. We choose 1ms as our fine granularity to reduce noise as in [24]. We generate coarse-grained time series by sampling (as described in §2.1) at 50ms granularity. Thus, the goal of the imputation methods is to zoom into the coarse-grained queue length by a factor of 50, *i.e.,* turn from 50ms granularity to 1ms.

Downstream tasks: Our tasks relate to bursts as they are hard to capture and critical for network operations. We identify bursts in both ground truth and imputed queue lengths using a specific method [56]. Subsequently, we calculate the normalized errors of burst occurrence, burst height, burst frequency, average inter-arrival time between consecutive bursts, and the number of queues experiencing a burst at the same 1ms interval during 10s. We also consider the frequency of empty queues which is crucial for queue health [23].

Results: Fig. 4 illustrates a representative example of a queue length on a short time period. When the IterativeImputer imputes it (Fig. 4a), it learns nothing from the auxiliary time series and simply connects periodic and maximum queue values. The transformer alone (Fig. 4b) detects the location of the burst but not its max and is thus inconsistent. The Transformer+KAL (Fig. 4c) learns to impute more consistent queue lengths, while Transformer+KAL+CEM (Fig. 4d) is forced to be consistent.

Table 1 summarizes our results on evaluating different versions for: (a-c) the queue imputation consistency constraints; (d-g) downstream tasks related to bursts; (h) queue health and (i) concurrent queue bursts. Lower values are better in all rows. The consistency errors (a-c) are mostly improved by incorporating KAL, and nullified by CEM. As KAL encourages higher values when bursts occur, the transformer can end up overshooting, leading to an increase in max-constraint error when only KAL is incorporated. This highlights the need for both CEM and KAL. The remaining tasks are improved by 11-96% when incorporating both KAL and CEM compared to transformer alone. We also observe that CEM does not always improve the performance of burst-related tasks. That is a trade-off between enforcing consistency and learning useful patterns. The IterativeImputer does 16-98% worse than Transformer+CEM+KAL across all tasks.

The average time for CEM to correct a 50ms transformer output is 1.47s, a significant improvement compared to FM alone which did not terminate for such imputations (§2.3).

5 Future Directions

Other means of integrating network knowledge: Our current system only uses constraints that are (or can be easily made) differentiable and thus can be incorporated into the transformer's loss. However, network constraints are often not differentiable with respect to learnable parameters. An interesting research direction for those constraints is to train an ML model to learn the functions that imply satisfaction of the property [12]. Taking one step further, the model could output intermediate variables representing physical quantities [51] that are easier to constrain and can then be used to calculate the final result. For example, instead of imputing queue lengths directly, we can impute port incoming rates using ML and get queue lengths by associating them with routing information. *Research Question: How to optimally incorporate network-domain knowledge into ML-based systems?*

Combining FM and ML in other networking problems: Many networking problems tackled separately by ML and FM could be prime candidates for combining the two. For instance, performance estimators like DeepQueueNet [58] or Mimicnet [61] can benefit from FM by bounding the delay predictions according to the shared buffer size. Moreover, generating adversarial examples for network protocols could leverage ML-based solutions [25, 33, 46] for discovering adversarial inputs and FM-based systems to ensure these inputs mirror real-world scenarios. Finally in generating synthetic traces [41, 53], one can use knowledge in the form of rules to transform existing or (synthetic) traces into new ones. *Research Question: How do we strive a balance between the accuracy of FM and the creativity of ML for networking problems?*

Towards practical network telemetry imputation: Our initial exploration shows the potential of software imputation of network telemetry as an alternative to hardware upgrades. Yet, our current system is quite limited in terms of: *(i)* target signal (queue length); *(ii)* imputation granularity (1ms from 50ms); and incorporated knowledge. Further investigation is needed to allow such a system to generalize to other settings and time series. Among other improvements, we believe making the system work under strict timing requirements would be particularly useful, as some tasks (such as performance-driven routing [5, 11, 26, 30], rate adaptation [28], and attack detection/prevention [36, 42]) drive real-time network activation and are hence subject to strict timing constraints. *Research Question: Can we make telemetry imputation generalize and/or work in real-time?*

6 Acknowledgement

This work was supported by the National Science Foundation (NSF) through Grant CNS-2319442, a Google Research Scholar Award, and a Princeton Innovation Fund Award.

References

[1] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 632–647.

[2] 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*.

[3] Maria Apostolaki, Vamsi Addanki, Manya Ghobadi, and Laurent Vanbever. 2021. FB: A flexible buffer management scheme for data center switches. *arXiv preprint arXiv:2105.10553* (2021).

[4] Maria Apostolaki, Cedric Maire, and Laurent Vanbever. 2021. Perimeter: A network-layer attack on the anonymity of cryptocurrencies. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*. Springer, 147–166.

[5] 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.

[6] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. 2019. Fab: Toward flow-aware buffer sharing on programmable switches. In *Proceedings of the 2019 Workshop on Buffer Sizing*. 1–6.

[7] Mina Tahmasbi Arashloo, Ryan Beckett, and Rachit Agarwal. 2023. Formal Methods for Network Performance Analysis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 645–661.

[8] Arista. 2016. Arista LANZ Overview. https://arista.com/assets/data/pdf/Whitepapers/Arista_LANZ_Overview_TechBulletin_0213.pdf.

[9] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. 2021. Toward formally verifying congestion control behavior. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 1–16.

[10] Clark Barrett and Cesare Tinelli. 2018. *Satisfiability modulo theories*. Springer.

[11] Henry Birge-Lee, Maria Apostolaki, and Jennifer Rexford. 2022. It takes two to tango: cooperative edge-to-edge routing. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 174–180.

[12] Thomas A Henzinger Mathias Lechner Chatterjee, Krishnendu and Dorde Zikelic. 2023. A learner-verifier framework for neural network controllers and certificates of stochastic systems. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*. 3–25.

[13] 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*. 15–29.

[14] 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.

[15] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008*. Springer, 337–340.

[16] 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.

[17] 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.

[18] 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.

[19] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 343–356.

[20] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2018. Netcomplete: Practical network-wide configuration synthesis with autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 579–594.

[21] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. 2021. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 305–318.

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

[23] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1, 4 (Aug 1993), 397–413.

[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] Tomer Gilad, Nathan H Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. 2019. Robustifying network protocols with adversarial examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. 85–92.

[26] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. 2019. Blink: Fast Connectivity Recovery Entirely in the Data Plane. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 161–176.

[27] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 15–26.

[28] Mehdi Hosseinzadeh, Karthick Shankar, Maria Apostolaki, Jay Ramachandran, Steven E Adams, Vyas Sekar, and Bruno Sinopoli. 2023. CANE: A Cascade Control Approach for Network-Assisted Video QoE Management. *IEEE Transactions on Control Systems Technology* (2023).

[29] 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.

[30] Kuo-Feng Hsu, Ryan Beckett, Ang Chen, Jennifer Rexford, and David Walker. 2020. Contra: A programmable system for performance-aware routing. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 701–721.

[31] Nathan Jay, Noga H Rotman, P Godfrey, Michael Schapira, and Aviv Tamar. 2018. Internet congestion control via deep reinforcement learning. *arXiv preprint arXiv:1810.03259* (2018).

[32] Alan Jeffrey and Taghrid Samak. 2009. Model checking firewall policy configurations. In *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 60–67.

[33] Samuel Jero, Md Endadul Hoque, David R Choffnes, Alan Mislove, and Cristina Nita-Rotaru. 2018. Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach.. In *NDSS*.

[34] Qiao Kang, Jiarong Xing, Yiming Qiu, and Ang Chen. 2021. Probabilistic profiling of stateful data planes for adversarial testing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 286–301.

[35] 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.

[36] Ege Cem Kirci, Maria Apostolaki, Roland Meier, Ankit Singla, and Laurent Vanbever. 2022. Mass surveillance of VoIP calls in the data plane. In *Proceedings of the Symposium on SDN Research*. 33–49.

[37] Ankit Kumar, Max von Hippel, Pete Manolios, and Cristina Nita-Rotaru. 2022. Formal Model-Driven Analysis of Resilience of GossipSub to Attacks from Misbehaving Peers. *arXiv preprint arXiv:2212.05197* (2022).

[38] John H Lagergren, John T Nardini, Ruth E Baker, Matthew J Simpson, and Kevin B Flores. 2020. Biologically-informed neural networks guide mechanistic modeling from sparse experimental data. *PLoS computational biology* 16, 12 (2020), e1008462.

[39] 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.

[40] 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)*.

[41] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. 2019. Generating high-fidelity, synthetic time series datasets with doppelganger. *arXiv preprint arXiv:1909.13403* (2019).

[42] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches. In *30th USENIX Security Symposium (USENIX Security 21)*. 3829–3846.

[43] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. 2020. Contention-aware performance prediction for virtualized network functions. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 270–282.

[44] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. 2019. Sizing Router Buffers (Redux). *SIGCOMM Comput. Commun. Rev.* 49, 5 (nov 2019), 69–74.

[45] NS3. 2023. NS3 Network Simulator. https://www.nsnam.org/.

[46] Anthony Peterson, Samuel Jero, Md Endadul Hoque, David R Choffnes, and Cristina Nita-Rotaru. 2020. aBBRate: Automating BBR Attack Exploration Using a Model-Based Approach. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. 225–240.

[47] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision* 40, 2 (2000), 99.

[48] Scikit-Learn. 2023. IterativeImputer¶. https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html.

[49] Rahul Anand Sharma, Ishan Sabane, Maria Apostolaki, Anthony Rowe, and Vyas Sekar. 2022. Lumen: a framework for developing and evaluating ML-based IoT network anomaly detection. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. 59–71.

[50] Kausik Subramanian, Loris D'Antoni, and Aditya Akella. 2017. Genesis: Synthesizing forwarding tables in multi-tenant networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. 572–585.

[51] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. 2020. Hamiltonian Generative Networks. (2020). arXiv:cs.LG/1909.13789

[52] 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

[53] 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.

[54] 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.

[55] Jackson Woodruff, Andrew W Moore, and Noa Zilberman. 2019. Measuring burstiness in data center applications. In *Proceedings of the 2019 Workshop on Buffer Sizing*. 1–6.

[56] 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.

[57] Zhiying Xu, Francis Y. Yan, Rachee Singh, Justin T. Chiu, Alexander M. Rush, and Minlan Yu. 2023. Teal: Learning-Accelerated Optimization of WAN Traffic Engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*. New York, NY, USA, 378–393.

[58] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. 2022. DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-Level Visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 441–457.

[59] 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.

[60] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*. 78–85.

[61] Qizhen Zhang, Kelvin KW Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. 2021. Mimicnet: fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 287–304.

[62] 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.

[63] Timothy Zhu, Daniel S Berger, and Mor Harchol-Balter. 2016. SNC-Meister: Admitting more tenants with tail latency SLOs. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 374–387.