

ML-assisted latency assignments in time-sensitive networking

Alexej Grigorjew, Michael Seufert, Nikolas Wehner, Jan Hofmann,
Tobias Hoßfeld

Angaben zur Veröffentlichung / Publication details:

Grigorjew, Alexej, Michael Seufert, Nikolas Wehner, Jan Hofmann, and Tobias Hoßfeld. 2021. "ML-assisted latency assignments in time-sensitive networking." In IFIP/IEEE International Symposium on Integrated Network Management (IM), 17-21 May 2021, Bordeaux, France, 116-24. Piscataway, NJ: IEEE.
<https://ieeexplore.ieee.org/document/9463924/>.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/>



ML-Assisted Latency Assignments in Time-Sensitive Networking

Alexej Grigorjew, Michael Seufert, Nikolas Wehner, Jan Hofmann, Tobias Hoßfeld

University of Würzburg, Germany

{alexej.grigorjew | michael.seufert | nikolas.wehner | tobias.hossfeld}@uni-wuerzburg.de, mail@hofmannjan.com

Abstract—Recent developments in industrial automation and in-vehicle communication have raised the requirements of real-time networking. Bus systems that were traditionally deployed in these fields cannot provide sufficient bandwidth and are now shifting towards Ethernet for their real-time communication needs. In this field, standardization efforts from the IEEE and the IETF have developed new data plane mechanisms such as shapers and schedulers, as well as control plane mechanisms such as reservation protocols to support their new requirements. However, their implementation and their optimal configuration remain an important factor for their efficiency. This work presents a machine learning framework that takes on the configuration task. Four different models are trained for the configuration of per-hop latency guarantees in a distributed resource reservation process and compared with respect to their real-time traffic capacity. The evaluation shows that all models provide good configurations for the provided scenarios, but more importantly, they represent a first step for a semi-automated configuration of parameters in Time-Sensitive Networking.

I. INTRODUCTION

Traditionally, real-time networks in the fields of industrial automation and in-vehicle communication are implemented by using bus systems or proprietary systems based on Ethernet. With careful admission control, it is possible to acquire latency guarantees for streams with low data rates if they adhere to typical periodic transmission patterns. However, with the recent advancements in these fields, the demand for higher data rates is increasing, while at the same time, more flexibility is required. Latency requirements become more diverse, systems are more dynamic with streams emerging and disappearing during run time, and network operators wish to use the same network infrastructure for both critical real-time traffic and bursty, non-reserved best-effort traffic simultaneously.

With these requirements in mind, the IEEE Time-Sensitive Networking (TSN) and the IETF Deterministic Networking (DetNet) working groups are developing a series of standards to support converged real-time networking with high data rates in traditional networking devices such as Ethernet switches. These standards include new shaping mechanisms, such as Credit-Based Shaping (CBS) [1] and Time Aware Shaping (TAS) [2], as well as advancements for admission control in the control plane, such as the Stream Reservation Protocol [3] [4], the Resource Reservation Protocol (RSVP-IntServ) [5], and the

Resource Allocation Protocol [6]. In the past, standardization groups have published profiles [7] that describe how to apply the standards for a specific use case, but their guarantees were proven to be inaccurate [8]. More recent profiles are under active development [9]–[12], but due to the increased demand for dynamic operation, they cannot provide an efficient network configuration for each situation. In general, the individual tools are well established by now, but the understanding of their optimal deployment and configuration remains an important research topic for time-critical communication.

Contribution: This work presents a machine learning (ML) framework for the configuration of real-time networks with a distributed reservation process. It implements a heuristic solution for an important optimization problem during real-time network planning: how should delay guarantees be assigned to queues such that the network utilization can be maximized? Therefore, multiple supervised learning and reinforcement learning models have been created and evaluated against an optimal brute force solution for two types of scenarios. The evaluation shows that, in the considered scenarios, most ML models attain roughly 90% of the brute force performance at a much lower computation time. These results can provide valuable input for the development of ML-assisted network configuration in TSN.

The remainder of this work is structured as follows. Related work regarding Time Sensitive Networking and machine learning for network configuration is covered in Section II. The methodology used for resource reservation simulation, ground truth generation, feature extraction, and ML model training is described in Section III. Section IV presents the results of the performance analysis of the ML-based heuristics in comparison with the optimal configuration. Finally, Section V concludes the paper and outlines future works.

II. RELATED WORK

A. Time Sensitive Networking

In order to meet real-time requirements and ensure a deterministic transmission behavior for time-sensitive traffic, the IEEE 802.1 working group proposed a set of standards under the term Time-Sensitive Networking (TSN [13]). TSN utilizes the Precision Time Protocol (PTP [14]) which synchronizes time information with sub-microsecond accuracy across all network participants. This is a prerequisite for mechanisms like the Time Aware Shaper (TAS [15]) which introduces timed

gates to each queue of a switch port. Mechanisms that do not rely on time synchronization include Credit-Based Shaping (CBS [16]) and Asynchronous Traffic Shaping (ATS [17] [18]), which enable reshaping of traffic in per-priority and per-stream granularity, respectively.

While each mechanism improves the latency guarantees of the data plane, bounded latency can already be guaranteed with Strict Priority Transmission Selection alone (SP [13]). Each switch manages up to eight queues dedicated to frames of different priority. SP does not require clock synchronization, and in contrast to CBS and ATS, it is widely supported by current hardware. SP was recently proven to provide deterministic guarantees even in a distributed plug-and-play reservation process [19] without the need for shapers. This work initially builds on the mechanism and attempts to configure the required latency guarantee thresholds for maximum utilization.

In addition to shapers, TSN specifies complementary mechanisms for additional reliability (FRER [20]) and network configuration. As this work is focused on distributed scenarios, it relies on mechanisms such as the Stream Reservation Protocol (SRP [3] [4]) or the new Resource Allocation Protocol (RAP [21]). The information provided by these protocols can be used by all switches to perform admission control and therefore guarantee bounded latency.

B. ML for Network Configuration

Machine learning (ML) describes the process of algorithmic learning from data. Due to its successful application to many real world problems, the research community has started to apply ML to the networking domain [22], where complex problems occur, in particular, with respect to network operations and management. While unsupervised learning is rarely used for network configuration [23], there are a few approaches which are based on supervised learning. Some of these works predict relevant metrics, which can then be used for network management. For example, [24] forecast traffic in order to provision virtual network functions, or [25] analyzed QoE factors from encrypted video traffic, which can be used for QoE-aware traffic management. Other works directly applied supervised learning to predict a network configuration, such as [26], which used a deep belief architecture to predict optimal routing paths. [27] compared different supervised learning methods to predict optimal locations of SDN controllers. The vast majority of related works on ML-based network configuration resorts to reinforcement learning (RL) [22], [28], as finding an optimal network configuration by interacting with a dynamic network environment nicely resembles the general problem statement of reinforcement learning.

One of the first attempts to apply RL to dynamic networks was proposed by [29] using the tabular Q-Learning algorithm in order to optimize packet routing. [30] was one of the first to apply Q-Learning to packet scheduling in routers. Both methods use a basic environment with a small state-action space and embed the RL module into the nodes of the switching network. As a more modern approach, [31] proposed an adaptive routing algorithm based on Q-Learning that took Quality of Service

(QoS) into consideration in the context of software-defined networking (SDN). [32] proposed a routing algorithm based on a Deep Q-Network (DQN) method that produced near-optimal routing configuration and minimized end-to-end delay in dynamic networks. [33] applied an Actor-Critic method to congestion control where actions represented changes to the traffic rate.

However, there are only few related works when it comes to applying ML in the context of dynamic real-time networks with deterministic guarantees. [34] employed a Markov chain Monte Carlo method using ML for low-latency routing. Moreover, [35] investigated the use of the k-nearest neighbors classification algorithm for assessing schedulability in TSN. Finally, there are conceptual works [36], [37] which envision ML for fault detection or self-configuration of TSN networks in the future. Our work provides a first step towards this vision by implementing a concrete ML decision model for the configuration of distributed TSN networks.

III. METHODOLOGY

The key problem that is addressed in this work is the configuration of per-hop latency guarantees C_p for each real-time traffic class p in the network. During the network's operation, new stream reservations toggle the admission control system. This system computes the current latency bounds d_p and compares them to the configured guarantees C_p to decide whether or not the new reservation can be accepted. The capacity N of the network denotes the number of accepted stream reservations. The task is to configure the latency guarantees C_p such that the capacity N is maximized.

This section presents the latency bound computation (III-A), the considered scenarios (III-B), the computation of the ground truth (III-C), the used features (III-D), and the implementation of supervised (III-E) and reinforcement learning (III-F).

A. Latency Bound Computation

The delay bound d_p of traffic class p is computed as explained in [19]:

$$d_p \leq \sum_{x \in \mathcal{H}_p} \frac{b_x z_x(C_x, C_p)}{r} + \sum_{x \in \mathcal{E}_p} \frac{b_x z_x(C_x, 0)}{r} + \max_{x \in \mathcal{L}_p} \frac{\hat{\ell}_x}{r} \quad (1)$$

Here, $z_x(C_x, C_p)$ is a function that returns the number of bursts from stream x that increase the queuing delay for the observed priority p . It depends on the configured guarantees C , on the data rate r_x of stream x , and on its path. The sets \mathcal{H}_p , \mathcal{E}_p , \mathcal{L}_p contain the streams from higher priorities, the same priority p , and lower priorities respectively. The denominator r is the link speed in bits/s, while b_x is the burst size and $\hat{\ell}_x$ the largest frame size of stream x . For further details, refer to [19].

B. Framework and Scenarios for Configuration

In the following, a framework for simulating and configuring basic industrial networks with different topologies is described. The framework is initialized with a topology, a set of streams \mathcal{S} , and a configuration \mathcal{C} . The main task of the framework is

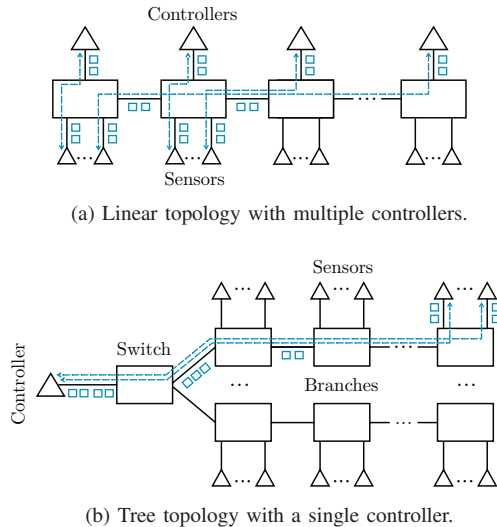


Fig. 1. Considered topologies for both training and evaluation.

to simulate the real-time communication between sensors and controllers and to determine the actual network capacity N . The capacity is defined as the number of streams $s \in \mathcal{S}$ that can be added to the network under the current configuration \mathcal{C} without violating the end-to-end latency requirements of any stream in the network.

The framework provides an interface that allows to adjust the current configuration \mathcal{C} and re-calculate the network capacity N under the new configuration \mathcal{C}' . Networks studied in this work are restricted to linear and tree topologies. A linear network topology is built upon a number of m linearly arranged switches. Each switch is connected to a controller and a number of n sensors. Each sensor is communicating with a random controller in the network. In contrast, tree topologies consist of a single top switch with a single controller. Every sensor communicates with this central controller, such that its link is the determining point of the network capacity. Without loss of generality, all links are 1 Gbit/s. Examples for a linear and a tree topology are depicted in Figure 1a and 1b, respectively.

The sensor-controller communication patterns are based on five application stream profiles, which are specified in Table I. Each profile includes a periodic transmission interval, an end-to-end latency requirement for frames, a minimum and maximum frame size, and a burst value that is equal to the maximum frame size. The properties of a stream $s \in \mathcal{S}$ are selected uniformly distributed from one of the application profiles.

Note that the models do not rely on periodic transmission, but the application profiles were chosen based on exemplary application update cycles from industrial automation. The topology layouts and sensor-controller communication patterns are based on real world industrial networks, adapted for increased network convergence as envisioned by TSN. They are primarily based on typical PROFINET use cases that synchronize the state of an industrial appliance periodically between sensors and their corresponding controller. These controllers are not to be confused with network controllers, they merely represent end devices in the presented use case.

TABLE I
AVAILABLE APPLICATION PROFILES FOR THE STREAMS.

	Transmission Interval	Maximum Latency	Minimum Frame Size	Maximum Frame Size
Profile 1	250 μ s	250 μ s	64 bytes	128 bytes
Profile 2	500 μ s	500 μ s	128 bytes	256 bytes
Profile 3	1000 μ s	1000 μ s	256 bytes	512 bytes
Profile 4	2000 μ s	2000 μ s	512 bytes	1024 bytes
Profile 5	4000 μ s	4000 μ s	1024 bytes	1522 bytes

No additional traffic shaping mechanism is used apart from basic SP transmission selection. Without loss of generality, up to four traffic classes are used for real-time traffic in these scenarios, while the other classes are reserved for different priorities of best-effort traffic. In general, it is possible to define a separate latency guarantee per traffic class for each individual egress port in the network, based on the respective load. In this stage of the work, the configuration is limited to a global configuration for all egress ports due to the implementation of the ground truth generation algorithm. This limitation may disappear in future versions of this work. The configuration of the network is defined as a 4-tuple $\mathcal{C} = (C_0, C_1, C_2, C_3)$ where C_p is the guaranteed maximum per-hop latency provided by priority class p . C_0 denotes the highest priority class with the lowest guaranteed per-hop latency. The priority $p(s)$ of a stream s is given by the lowest priority class that satisfies the per-hop requirement $\delta^h(s)$ at the respective hop h . If there is no priority class that meets this requirement, s is not assigned a priority and cannot be added to the network under the current configuration.

As all ports are configured equally, the per-hop latency requirement for s can be calculated with the maximum end-to-end latency $\delta(s)$ and the number of hops $h(s)$ on the stream's path: $\delta^h(s) = \delta(s)/h(s)$.

C. Ground Truth Generation

During supervised learning and for the evaluation of the model's performance, it is important to know the ground truth for all generated scenarios. As the task of the ML model is to solve an optimization problem, the ground truth is given by the optimal solution that allows the reservation process to accept the maximum number of streams in the network.

Albeit not the same, the choice of latency guarantees C_i is related to the bin packing problem where the cost function of each individual item depends on the chosen bin and on its latency configuration. Increasing an individual latency guarantee C_i may increase the number of streams that can be accepted for that traffic class, but it may also become too high for some streams, causing them to shift towards the higher priority and increase resource usage there. Additionally, in the advanced version of this problem, different ports in the network may be given different priorities in order to allow for a relaxed configuration at bottlenecks, while compensating with tighter latency guarantees in other parts of the network to achieve the same end-to-end latency. In this work, the simple version of this optimization problem is assumed where all egress ports share the same latency guarantee configuration.

In general, this can be solved by an integer linear program that includes constraints regarding path consistency, bandwidth utilization, and maximum end-to-end latency. For each pair of individual stream s and traffic class p , a binary variable $prio[s, p] \in \{0, 1\}$ indicates whether stream s belongs to priority level p , with $\sum_{p=0}^3 prio[s, p] \leq 1$. In addition, a bounded variable $conf[p] \in \mathbb{R}$ can be used for the latency guarantee configuration. The term $\sum_{p=0}^3 prio[s, p] \cdot conf[p]$ can then be employed as the stream's current delay in the latency constraint. The goal is to maximize the number of successfully placed streams, i.e., $\max \sum_{s \in \mathcal{S}} \sum_{p=0}^3 prio[s, p]$. This formulation has two major limitations: (1) It quickly reaches the limits of available hardware resources during computation, both regarding runtime and memory consumption. (2) In real deployments, it may not be possible to choose whether or not to accept a specific stream. Stream reservations are issued in random order with no way to predict future reservations.

Therefore, for the current version of this work, the order of stream reservations is fixed in each scenario. As each port is configured equally, the per-hop latency requirement $\delta^h(s)$ can be computed from each stream's end-to-end requirement and hop count: $\delta^h(s) = \delta(s)/h(s)$. A brute force algorithm is applied that computes every possible choice $\mathcal{P} = \{\delta^h(s) \mid s \in \mathcal{S}\}$ for individual per-hop guarantees C_i and tests every combination of these choices for the four considered priority levels. This is equivalent to drawing four individual numbers out of $|\mathcal{P}|$ possibilities, hence the algorithm must test $\binom{|\mathcal{P}|}{4} = \frac{|\mathcal{P}|!}{4! \cdot (|\mathcal{P}|-4)!}$ configurations. During each test, a simulation of consecutive resource reservations for a given topology and a given list of streams is conducted, and the configuration with the maximum number of accepted streams is reported. This value is used as ground truth for supervised learning, and for comparison during performance evaluation of the ML-based solutions.

For 5 application profiles (Table I), up to 600 streams, and a maximum path length of 7, the brute force algorithm finds the optimal solution within at most a few minutes. The algorithm scales polynomially with the number of streams, and exponentially with the number of application profiles and the maximum path length. Slightly increasing these values already leads to several hours or days of execution time. The anticipated industrial networks are expected to have higher path lengths and more heterogeneous communication patterns. This is why an efficient heuristic is necessary for large-scale decision making.

D. Static and Dynamic Features

Both the supervised learning and the reinforcement learning models rely on a thorough set of features, representing as much of the scenario as possible. The training and test scenarios are comprised of a specific network topology and a set of streams, which must be represented by a fixed-length vector of features. This is a challenging task, as networks differ in size, and the number of streams is not always the same. For this reason, whenever there is a feature that refers to a specific switch, port, stream, or anything with variable number of occurrence in general, statistical properties of all such entities are used

instead. Each such characteristic is summarized by the following properties: minimum, maximum, mean, standard deviation, variance, coefficient of variation, skewness, and kurtosis. In addition, a probability density function of such properties can be estimated by a histogram with 10-20 bins, which are carefully chosen to include the majority of reasonable values considering different scenarios for each property. The characteristics for the static feature vector include 520 individual values:

- (i) topology information: node degrees, betweenness centrality, closeness centrality,
- (ii) stream quantities: number of streams, path lengths,
- (iii) traffic specifications: data rates, frame sizes, bursts,
- (iv) aggregated per-port data rates and bursts,
- (v) the results of Eq. 1 for a hypothetical baseline configuration with 20 priority levels, exponentially spaced between 100 ns and 128 ms.

In particular, number (iv) refers to aggregated traffic volume depending on the requested maximum latency, based on pre-configured ranges. For example, the data rates of all streams whose latency is within $[250 \mu\text{s}, 500 \mu\text{s}]$ would be accumulated for each port. Number (v) is a measure of load. A higher latency bound in this hypothetical configuration indicates a higher load for the latency range of the respective priority level.

Dynamic features for reinforcement learning. Reinforcement learning models apply an iterative process and change the current situation in the network via actions. These actions include a slight increase or decrease of individual latency guarantees. Based on the feedback from the environment, i.e., whether there could be more or less streams placed in the network afterwards, these actions should push the configuration towards the optimal value. As this is a dynamic scenario with a varying situation, a few dynamic features are included to support the decision process.

Dynamic features include the following characteristics related to the current situation:

- (i) the result of the latency bound formula (Eq. 1) based on the four currently selected guarantees \mathcal{C} for each port and each priority in the network,
- (ii) for each priority, the number of streams that could successfully be deployed,
- (iii) the total number of successfully deployed streams,
- (iv) the fraction of currently used bandwidth on each link,
- (v) the amount of streams that cannot conduct stream reservations because their latency requirements are higher than the highest priority's guarantee.

Those characteristics that are computed for each port in the network (i and iv) are once again statistically summarized to have a fixed width output for the feature vector.

E. Supervised Learning

1) *Random Forest Classifier:* The first supervised ML approach is a random forest classifier implemented in scikit-learn. The generated ground truth is used to learn a representation, which estimates the four priority classes by selecting one of

TABLE II
FINAL HYPERPARAMETERS FOR ALL METHODS.

Method	Hyperparameters
Classification	Features=All, Aggregator=Mean, Trees=30, Max. Tree Depth=10
Regression	Layer1=1024, Layer2=1024, Layer3=1024, Batch Size=256
DQN	Epochs=18, Episodes=50, Layer1=110, Layer2=99, Target Update=450, Batch Size=250, $\alpha=0.0005$, $\gamma=0.8$
Actor-Critic	Epochs=16, Episodes=70, Layer1=96, Layer2=86, $\alpha=0.075$, $\gamma=0.88$

20 pre-defined bins. These bins shall represent optimal latency bounds and range evenly spaced from 0 to 1000 μs .

The data is split into 80% training samples and 20% test samples. The four priority classes of the ground truth serve as target classes. These target classes are balanced within the training samples using bootstrapping and all data are scaled between 0 and 1. The metric for evaluating the quality of the model’s predictions is a custom accuracy metric, which computes the accuracy between the four priority classes and the four ground truth priority classes element-wise and returns the minimum observed accuracy of all element-wise comparisons. Finally, the hyperparameters are optimized using a grid search and 3-fold cross-validation. The final tunable hyperparameters for the random forest classification are shown in Table II and include the dimensionality of the features (all 520 static features), the number of trees in the forest (30), and the maximum depth of the trees (10).

2) *Neural Network Regressor*: A neural network performing a regression task is the second tested supervised ML approach. Based on the input features, the neural network is supposed to directly estimate the four priority classes.

For training of the neural network, the data is split into a training set, validation set, and test set, where the shares of the data for the sets are 60%, 20%, and 20%. Like before, no feature selection is performed for this approach, so all 520 static features are used, the target priority classes are balanced using bootstrapping, and the features are normalized.

The architecture of the neural network consists of three hidden layers with 1024 neurons each followed by a dropout layer with a rate of 20%. All hidden layers use *tanh* as activation function. The output layer uses four nodes, representing the estimated priority classes, which are activated with the *sigmoid* function. The number of nodes in the input layer correspond to the input feature dimension. The neural network is implemented with Keras and a Tensorflow backend.

The well-known Adam optimizer is used and the loss is calculated with the mean squared error (MSE). The metrics for evaluating the quality of the model’s predictions are further the mean absolute error (MAE), the mean absolute percentage error (MAPE), and the cosine similarity. A batch size of 256 and a maximum of 1000 epochs are used for learning. Additionally, an early stopping mechanism with a patience of 50 is implemented and the best model, i.e., the model with the lowest observed loss, is stored.

F. Reinforcement Learning

1) *Environment*: Reinforcement Learning is expected to be an appropriate method that approaches a decent configuration in

reasonable time. Therefore, this section proposes an interactive environment built upon the framework that allows to apply different RL methods to the problem.

The essential components of such an environment are the action space, the state space, and the reward function. As the objective is to identify an optimal configuration C^* , actions represent adjustments to the configuration. The action space is defined as $\mathcal{A} = (a_0, a_1, \dots, a_{2c-1})$, where c is the number of classes that are used by the framework. For every priority class, the environment provides one action to increase and one action to decrease the guaranteed per-hop latency of the class by 10 μs , respectively. To be precise, an action does not adjust the absolute value, but rather the absolute distance between class C_i and the next lower priority class C_{i-1} . This results in an adjustment of $\pm 10 \mu\text{s}$ to the class C_i itself as well as all higher priority classes. If the absolute distance between two classes is less than 10 μs , an action that would further decrease the distance between both classes results in a no-op action. In general, no-op actions are actions that are invalid in the current state and therefore do not affect the environment.

Altogether, the state is represented by a vector of 76 selected features from both the static and dynamic set. The static features include topology-specific features like the number of sensors, controllers, and switches, and the network diameter. Further, link-specific features like the mean link speed and stream-specific features like the total number of streams in the network are considered. In addition to these, a hypothetical static example configuration with eight priorities is used to compute bounds that represent the relative load of various latency ranges between 25 μs and 10 ms.

In contrast to the supervised ML approaches, RL allows to incorporate dynamic features which are updated after each action in the environment. The RL agent requires a reward signal which allows the agent to learn the desired behavior. Therefore, the reward function implements the optimization objective which is to maximize the network capacity N . Equation 2 defines a reward function that corresponds to the percentage increase in capacity after adjusting the network configuration, where $|\mathcal{S}|$ is the total number of streams and N_t is the number of accepted streams after the t -th action.

$$r_t = \begin{cases} \frac{N_t - N_{t-1}}{|\mathcal{S}|} \cdot 120 & \text{if } N_t = |\mathcal{S}| \\ \frac{N_t - N_{t-1}}{|\mathcal{S}|} \cdot 100 & \text{otherwise} \end{cases} \quad (2)$$

If the agent can find an optimal configuration with $N_t = |\mathcal{S}|$, it is rewarded by an additional 20%. The environment provides immediate rewards, the agent receives a reward signal after every single action rather than at the end of an episode.

In RL, an episode of actions normally ends when a terminal state is reached. But as the problem studied in this work is an optimization problem, it is not trivial to determine if a state is actually the terminal state with the best possible configuration for a specific scenario. The terminal state is only obviously optimal for the configuration $N_t = |\mathcal{S}|$. For this reason, an early stopping mechanism is applied, which prevents episodes

from continuing endlessly in case that it is impossible to reach such an optimal terminal state.

At the beginning of an episode, framework and environment are initialized with a topology and a set of streams \mathcal{S} . The initial configuration of the network is not chosen randomly, but rather as a rough estimation of a configuration that would match the given set of streams. This initial choice deserves some consideration, because it can significantly speed up the training process if there is already a reasonable initial network configuration. For such a configuration, the actual occurring per-hop latency requirements for all streams are consulted. Let $\Delta = (\delta^h(s) \mid s \in \mathcal{S})$ define an ordered list of all uniquely occurring per-hop requirements. The initial network configuration \mathcal{C} , for p classes is then determined for class C_0 by using the lowest per-hop requirement of Δ , i.e., the first element of Δ , and for the classes $C_{i=1\dots c-1}$ by using the $\text{floor}(100 \cdot \frac{i}{c})$ percentiles of Δ . For $c = 4$ configured guarantees, the first element, the 25th percentile, the 50th percentile, and the 75th percentile of Δ are chosen for the initial configuration.

Industrial networks are highly dynamic environments. Communication varies in that endpoints exchange frames of different sizes at different rates. Also, the structure and size of the network can vary in that new connections are established or new endpoints are added to the network. To account for this, the agent is trained on a number of different network topologies, and for each such topology, different sets of streams are generated, each referred to as a scenario.

2) *Deep Q-Network*: The first applied RL method is the Deep Q-Network (DQN), which also serves as a representative for the group of off-policy learning methods. As an extension to the well-studied Q-Learning method it has already been successfully applied to various problems like video game control [38] or robotic control [39].

The input of the Q-Network corresponds to the state vector discussed above. The size of the input and output layer correspond to the size of the state vector and the size of the action space, respectively. ReLU activation is applied to the two hidden layers and no activation is applied to the output layer. This allows for negative output values, which is desirable for the Q-Network as it approximates action-values. The values estimate the cumulative future reward when taking an action and represent the quality of the action.

According to [38], a target network should be used with an architecture identical to the online network. The target network is utilized to approximate the target value that is then used to calculate the loss. The loss is computed with the mean squared error (MSE). Based on the loss, an optimizer is used to compute the gradient, which is then back-propagated through the target network. The number of time steps after which the target network is synchronized with the online network is regarded as a tunable hyperparameter.

Methods like DQN require some exploration strategy like ε -greedy. The idea is that, especially in early phases of the training, the agent performs random actions with probability ε in order to discover valuable state-action pairs. To account

for changing topologies and scenarios, the ε -greedy strategy is adjusted so that exploration and exploitation are balanced over all topologies and scenarios.

The hyperparameters are optimized using a fractional factorial design [40]. The tunable parameters include the number of epochs, the number of episodes, the learning rate, the discount factor, the optimizer, the hidden nodes, the target update rate, and the batch size. Table II presents the final value for each hyperparameter. With a smaller number of input features, the number of neurons per layer also needs to be decreased. In contrast to the regression approach with more input features and 1024 neurons per layer, the DQN layers contain only 110 and 99 neurons, respectively. The target network and the online network are synchronized after 450 time steps.

3) *Actor-Critic*: The second RL method implemented in this work is Actor-Critic, which serves as a representative for the group of on-policy learning methods. A key difference to the DQN implementation is the network architecture. While DQN uses the Q-Network to approximate a single action-value function, Actor-Critic utilizes both an actor network to approximate a policy and a critic network that approximates a state-value function. In general, the term *actor-critic* refers to a group of algorithms and does not make an assumption about the actual implementation of the actor and the critic.

This work implements Actor-Critic using a single shared network for feature extraction and a split output layer for both actor and critic. The approach is easy to implement, reduces the number of trainable network parameters, and also ensures better comparability since the network architecture resembles that of DQN for the most part. Therefore, different results can be attributed to the different methods of learning rather than the different number of learnable parameters.

The size of the input layer, again, corresponds to the size of the state vector. As with DQN, the network uses two hidden layers. The output layer of the actor has a size equal to the size of the action space while the output layer of the critic has a fixed size of 1 and outputs a single state-value. ReLU activation is applied to both hidden layers. The split output layer, however, requires more consideration. For the critic output layer, no activation is applied, because the output is a state-value which can either be positive or negative based on the expected cumulative reward. For the actor, however, the output values correspond to a policy, which is a probability distribution. For this reason, *softmax* activation is applied. This ensures that output values are non-negative values within the interval $[0, 1]$ and that all output values sum up to 1. This results in the desired probability distribution which represents the current policy.

For the purpose of learning, a temporal-difference error is calculated at each time step. The gradient is computed from the sum of the actor loss and the critic loss and is then back-propagated through the shared network. This results in a simultaneous optimization of the policy and the accuracy of the state-value.

Overall, the Actor-Critic method used in this work is more

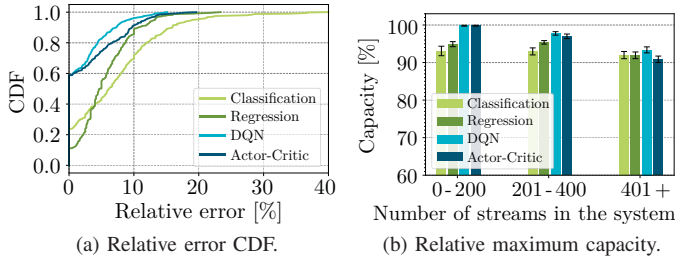


Fig. 2. Results for linear topologies.

straightforward than the DQN method as it involves less implementation details and therefore introduces less tunable hyperparameters. The tunable hyperparameters involve the number of epochs, the number of episodes, the neurons of the hidden layers, and the learning rate. These hyperparameters are again optimized using a fractional factorial design and the final values can be seen in Table II. Compared to DQN, the Actor-Critic network is slightly smaller with respect to the number of neurons per layer. Further, the learning rate α is significantly higher (0.075) and less epochs (16), but more episodes (70) are required for training.

IV. RESULTS

This work utilizes two different data sets for training both ML and RL models. The first one is a set of unique linear network topologies with up to 10 switches and up to 50 sensors per switch. The second data set includes unique tree topologies with up to three branches, up to 10 switches per branch and 50 sensors per switch. In order to reduce computational cost, topologies with more than 300 sensors are removed from both data sets. As each sensor communicates with one controller in a bidirectional manner, this makes for network topologies with a maximum of 600 streams in the system. For evaluation, 400 scenarios are chosen from the linear set and the tree set, respectively. It is assured that the test scenarios are not used for training the models.

A. Linear Topology

First, the results for the scenarios with linear topologies are presented. Figure 2a depicts a CDF for the differences of the accepted number of streams of a model and the number of accepted streams by the brute force algorithm for all test scenarios and models, i.e., the relative error between the ground truth and the model predictions is shown. The x-axis depicts the error in percentage relative to the ground truth and the y-axis denotes the fraction of scenarios observing such an error. The light green line represents the results for the random forest classifier and the dark green line shows the results for the neural network regression. Further, the results for the DQN model and for the Actor-Critic model are shown in light blue and dark blue, respectively. It can be observed that the RL models are able to place the same number of streams as the ground truth for approx. 60% of the linear topology scenarios, while the supervised learning techniques are only able to achieve this for around 24% of the scenarios (classification). Further, the

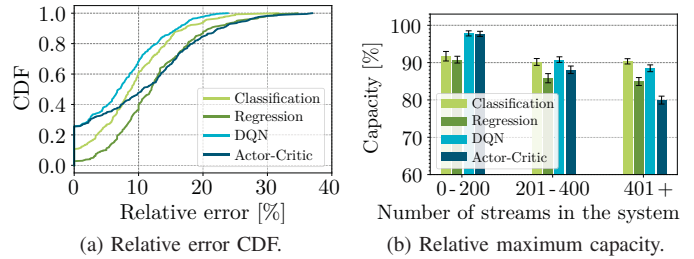


Fig. 3. Results for tree topologies.

maximum relative error is in particular for the RL methods with approx. 14% (DQN) and 19% (Actor-Critic) low. Here, the random forest classification provides the worst results as it shows maximum relative errors of up to 40%. In general, these results already indicate a superiority of reinforcement learning methods over supervised techniques for TSN configuration.

To investigate the impact of the network size on the performance of the RL and supervised models, Figure 2b illustrates the performance of the models with respect to different network sizes, here expressed by the number of streams in the system. The figure presents the average percentage of placed streams relative to the ground truth along with the 95% confidence intervals in dependency of the number of streams in the system. The x-axis, therefore, depicts the number of streams in the system, while the y-axis shows the percentage of placed streams relative to the ground truth, i.e., the network capacity. The bars are colored according to the models and in alignment with Figure 2a. The figure reveals that the RL methods are especially effective for small scenarios with less than 200 streams in the system. Here, the RL models perform identical to the brute force algorithm and are able to place all streams. On the other hand, the supervised techniques achieve only an average network capacity of approx. 92%. For more than 200 streams and less than 400 streams, the RL methods still outperform the supervised techniques, however, they are no longer able to provide the optimal results. Further, DQN starts to perform slightly better than Actor-Critic, while the supervised techniques perform similar to before. For more than 400 streams, the performance of Actor-Critic drops significantly, while DQN again shows the best performance of all models. The regression and classification approaches perform similar, but slightly worse compared to the smaller scenarios. Still, all presented models are able to place more than 90% of the streams compared to the ground truth even for the large scenarios.

When averaging the results over all linear topology scenarios, it can be summarized that the DQN model provides the best results with a 97.9% network capacity, followed by the Actor-Critic model with 97.1%. The regression approach achieved a network capacity of 94.5%, while the random forest classifier could place only 92.8% of placeable streams.

B. Tree Topology

Next, the performance of the models is evaluated when confronted with tree topologies. In contrast to linear topologies, tree topologies contain a switch over which all traffic has to be

exchanged and which serves as a bottleneck in the network. The reasonable configuration of this bottleneck switch complicates the configuration task slightly.

Figure 3a resembles Figure 2a and shows the resulting error CDFs for all models on the tree topology test set. It is visible that all methods perform significantly worse for the tree topologies when compared to the results of the linear topologies. Now, the RL models provide the optimal solution for approx. 25% of the scenarios, while the supervised models can provide an optimal solution for only 10% of the scenarios (classification) or even less (regression). Further, the maximum relative errors increased for most models. The lowest maximum relative error is again shown by DQN with approx. 23%, followed by the classification and regression approach with approx. 30% and 35%, respectively. The highest maximum relative error is this time shown by Actor-Critic (38%).

As with the linear topologies, Figure 3b investigates the impact of the network size on the the performance of the RL and supervised models for the tree scenarios.

The figure reveals that the RL methods are especially effective for small scenarios with less than 200 streams. Here, the RL models perform identical to the brute force algorithm and are able to place all streams. On the other hand, the supervised techniques achieve only an average network capacity of approx. 92%. For more than 200 streams and less than 400 streams, the RL methods still outperform the supervised techniques. However, they are no longer able to provide the optimal results. With respect to the supervised models, it can be stated that the random forest classifier outperformed the regression approach notably for the medium and large networks, while the performance for small networks can be considered similar. The RL methods perform significantly better than the supervised methods for small networks. The difference between RL and supervised models gets smaller for medium-sized networks. As with linear topologies, the performance of Actor-Critic drops significantly below the performance of all the other models for large networks. Here, the classification approach provides even slightly better results than the DQN model.

Again, DQN outperforms all other models with an average network capacity of 92.8%. In contrast to linear topologies, the random forest classifier (90.8%), however, surpasses the performance of the Actor-Critic model (89.3%) this time and the worst results are shown by the neural network regressor with an average network capacity of 87.4%.

C. Lessons Learnt

In summary, the presented results proved the successful application of RL and supervised learning techniques to the problem of SP configuration for linear and tree industrial real-time networks. RL provided optimal results for small linear networks and was able to outperform the supervised ML models on small and medium-sized linear networks. For linear networks, all models were able to provide capacities of 92%-97%. For the slightly more complex tree topologies, the performance of all models dropped significantly. In particular, the Actor-Critic model performed significantly worse on large

tree networks compared to the other models. The obtained capacities in general ranged between 87% and 93%.

Nonetheless, Actor-Critic exhibited the advantage that the model training took only 28% of the time required for the training of the DQN model. This can be explained by the lack of batch learning from memory for Actor-Critic. This advantage makes Actor-Critic the more suitable model in the field. In general, all models outperformed the brute force algorithm with respect to the required configuration time significantly. Once trained, the execution time of the ML heuristics is reduced to a few seconds compared to multiple hours or days required for the exhaustive algorithm. This emphasizes the fact that ML models provide a more economic solution for TSN configuration. Finally, the DQN model was able to outperform all other tested methods on both the linear and tree test sets and thus proved to be the best choice for dynamic SP configuration in linear and tree networks.

V. CONCLUSION AND OUTLOOK

This paper presented a tool set for machine learning (ML) assisted configuration of a real-time network with per-hop latency guarantees. It applied four ML models to configure the latency guarantee of each real-time traffic class such that the number of accepted streams during resource reservation is maximized. The training and evaluation scenarios include linear and tree-like topologies, focusing on industrial scenarios with sensor-controller communication patterns. The performance comparison indicates an accuracy of roughly 90% for the evaluated scenarios, with slightly better results obtained for the linear topologies. These results show that the methodology is an applicable approach towards efficient configuration of industrial real-time networks.

The current implementation can be extended in various ways. First, the training scenarios can be extended to include more use cases and optimization targets. For example, the current scenarios maximize the total number of accepted streams, without distinguishing the importance of individual streams, e.g., based on their data rate. Second, the latency configuration can be generalized further. Currently, each port is configured equally for applicability reasons. This can be extended such that every port receives its own configuration. In this case, streams cannot be assigned to priorities automatically, and further decisions must be made either during scenario definition or by the ML model. Further, future work may include more TSN shapers and take more decisions, such as selecting which shaper should be used in the first place, and providing additional configuration options for these mechanisms, such as idle slopes for the Credit-Based Shaper. Finally, the feature set of the current ML model assumes perfect knowledge of the later deployment. Oftentimes, network topology and especially stream information is not fully available during network planning, and decisions must be made with rough estimates of expected loads. It could be challenging to provide configuration guidelines without having specific scenarios in mind, especially regarding the reinforcement learning approaches that rely on the environment's feedback for their decisions.

REFERENCES

- [1] "IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks – Amendment: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2010*, 2010.
- [2] "IEEE Standard for Local and Metropolitan Area Network – Bridges and Bridged Networks – Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015*, 2015.
- [3] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat*, 2010.
- [4] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," *IEEE Std 802.1Qcc*, 2018.
- [5] J. T. Wroclawski, "The Use of RSVP with IETF Integrated Services," RFC 2210, Sep. 1997. [Online]. Available: <https://rfc-editor.org/rfc/rfc2210.txt>
- [6] "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment: Resource Allocation Protocol (RAP)," *IEEE Std P802.1Qdd*, 2019.
- [7] "IEEE Standard for Local and metropolitan area networks – Audio Video Bridging (AVB) Systems," *IEEE Std 802.1BA-2011*, pp. 1–45, 2011.
- [8] C. Boiger, "Class A bridge latency calculations," in *IEEE 802 November Plenary Meeting*, 2010.
- [9] "Time-Sensitive Networking Profile for Industrial Automation," *IEC/IEEE 60802/D1.2*, 2020.
- [10] "Time-Sensitive Networking for Fronthaul," *IEEE P802.1CM/D2.2*, 2020.
- [11] "Draft Standard for Local and Metropolitan Area Networks — Time-Sensitive Networking Profile for Service Provider Networks," *IEEE P802.1DF*, 2020.
- [12] "Draft Standard for Local and Metropolitan Area Networks — Time-Sensitive Networking Profile for Automotive In-Vehicle Ethernet Communications," *IEEE P802.1DG/D1.2*, 2020.
- [13] IEEE 802.1Q, "IEEE Standard for Local and Metropolitan Area Network – Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, 2018.
- [14] IEEE 1588, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 2008.
- [15] IEEE 802.1Qbv, "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014)*, 2016.
- [16] IEEE 802.1Qav, "IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks – Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, 2009.
- [17] IEEE 802.1Qcr, "IEEE Draft Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment: Asynchronous Traffic Shaping," *IEEE P802.1Qcr/D2.1, February 2020*, 2020.
- [18] J. Specht and S. Samii, "Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 75–85.
- [19] A. Grigorjew, F. Metzger, T. Hoßfeld, J. Specht, F.-J. Götz, F. Chen, and J. Schmitt, "Bounded latency with bridge-local stream reservation and strict priority queuing," in *2020 11th International Conference on Networks of the Future (NoF)*. IEEE, 2020.
- [20] IEEE 802.1CB, "IEEE Standard for Local and Metropolitan Area Networks – Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, 2017.
- [21] IEEE 802.1Qdd, "IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment: Resource Allocation Protocol," 2018.
- [22] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.
- [23] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges," *IEEE Access*, vol. 7, pp. 65 579–65 615, 2019.
- [24] A. Scalingi, F. Esposito, W. Muhammad, and A. Pescapé, "Scalable Provisioning of Virtual Network Functions via Supervised Learning," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 423–431.
- [25] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Stream-based Machine Learning for Real-time QoE Analysis of Encrypted Video Streaming Traffic," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2019, pp. 76–81.
- [26] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "Routing or Computing? The Paradigm Shift towards Intelligent Computer Network Packet Transmission based on Deep Learning," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1946–1960, 2017.
- [27] M. He, P. Kalmbach, A. Blenk, W. Kellerer, and S. Schmid, "Algorithm-data Driven Optimization of Adaptive Communication Networks," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–6.
- [28] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.
- [29] J. A. Boyan and M. L. Littman, "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Al-spector, Eds. Morgan-Kaufmann, 1994, pp. 671–678.
- [30] H. Ferrá, K. Lau, C. Leckie, and A. Tang, "Applying Reinforcement Learning to Packet Scheduling in Routers." 01 2003, pp. 79–84.
- [31] S. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 25–33.
- [32] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization," *CoRR*, vol. abs/1709.07080, 2017. [Online]. Available: <http://arxiv.org/abs/1709.07080>
- [33] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3050–3059.
- [34] Y. Nakayama, D. Hisano, T. Kubo, T. Shimizu, H. Nakamura, J. Terada, and A. Otaka, "Low-latency Routing for Fronthaul Network: A Monte Carlo Machine Learning Approach," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.
- [35] T. L. Mai, N. Navet, and J. Migge, "On the Use of Supervised Machine Learning for Assessing Schedulability: Application to Ethernet TSN," in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, 2019, pp. 143–153.
- [36] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN Networks," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2017, pp. 1–8.
- [37] N. Desai and S. Punnekkat, "Enhancing Fault Detection in Time Sensitive Networks using Machine Learning," in *2020 International Conference on Communication Systems & NETWORKS (COMSNETS)*. IEEE, 2020, pp. 714–719.
- [38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: <http://arxiv.org/abs/1312.5602>
- [39] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," *CoRR*, 2016. [Online]. Available: <http://arxiv.org/abs/1603.00748>
- [40] R. Myers, D. Montgomery, and C. Anderson-Cook, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, 01 2016, vol. 705.