Learning-based Incast Performance Inference in Software-Defined Data Centers

Kokouvi Benoit Nougnanke, Yann Labit, Marc Bruyère, Simone Ferlin, Ulrich Aivodji

Learning-based Incast Performance Inference in Software-Defined Data Centers

Kokouvi Benoit Nougnanke*, Yann Labit*, Marc Bruyere†, Simone Ferlin‡, Ulrich Aïvodji§
* *LAAS-CNRS, Université de Toulouse, CNRS, UPS, F-31400 Toulouse, France*
† *IIJ Innovation Institute, University of Tokyo, Tokyo, Japan*
‡ *Ericsson Research*
§ *Université du Québéc à Montréal*
nougnanke@laas.fr, ylabit@laas.fr, marc@iij.ad.jp, simone.ferlin@ericsson.com, aivodji.ulrich@uqam.ca

***Abstract*—Incast traffic is a many-to-one communication pattern used in many applications, including distributed storage, web-search with partition/aggregation design pattern, and MapReduce, commonly in data centers. It is generally composed of short-lived flows that may be queued behind large flows' packets in congested switches where performance degradation is observed. Smart buffering at the switch level is sensed to mitigate this issue by automatically and dynamically adapting to traffic conditions changes in the highly dynamic data center environment. But for this dynamic and smart buffer management to become effectively beneficial for all the traffic, and especially for incast the most critical one, incast performance models that provide insights on how various factors affect it are needed. The literature lacks these types of models. The existing ones are analytical models, which are either tightly coupled with a particular protocol version or specific to certain empirical data. Motivated by this observation, we propose a machine-learning-based incast performance inference. With this prediction capability, smart buffering scheme or other QoS optimization algorithms could anticipate and efficiently optimize system parameters adjustment to achieve optimal performance. Since applying machine learning to networks managed in a distributed fashion is hard, the prediction mechanism will be deployed on an SDN control plane. We could then take advantage of SDN's centralized global view, its telemetry capabilities, and its management flexibility.**

***Index Terms*—TCP Incast, Datacenters, SDN, Machine Learning, Performance Prediction, QoS**

I. Introduction

Datacenter workloads are composed essentially of long-lived flows or elephant flows (e.g., backup, replication, data mining) and short flows or mice flows (e.g., delivering search results). Besides this classification, datacenter workloads often require sending requests to large numbers of servers and then handling their near-simultaneous responses, causing a problem called incast [1]. This many-to-one communication and its associated traffic pattern are also called incast.

This many-to-one pattern in data centers is used for applications such as distributed storage (e.g., BigTable, HDFS, and GFS), web-search with partition/aggregation design pattern, and cluster computing platforms (e.g., MapReduce, Spark) [2]. Depending on the size of the servers' responses, we can distinguish between long-lived incast and short-lived incast. But it is worth mentioning that incast generally manifests in the short-lived form [3].

Incast traffic could cause severe congestion in switches and result in TCP throughput collapse, substantially degrading the application performance. The catastrophic TCP throughput collapse is explained by the fact that the bottleneck switch buffer is overfilled quickly as the number of competing senders increases. This leads to packet losses and subsequent retransmissions, after timeouts [3]–[5]. The TCP retransmission timeout (RTO) is computed dynamically based on experienced RTTs, but it is subject to a configuration minimum RTO, RTO_{min} of around hundred of milliseconds (e.g., 200ms). This is orders of magnitude too large for data center environments where RTT is in the 10s or 100s of microseconds.

This challenging traffic pattern handling is critical for Datacenters. Several solutions were proposed to mitigate the TCP throughput collapse in the incast scenario. Most of them concern RTO_{min} tuning to adequate small values in the RTT scale [4]–[6]. Another approach consists of using Explicit Congestion Notification (ECN) marking at the bottleneck switch level to ensure that senders are quickly notified of the queue overshoot and then adjusting their sending rate accordingly [7]. This prevents buffer overflow and subsequent timeouts. The work in [8] proposes an intelligent selective packet discarding at the switch level. This intelligent discarding ensures that the sender responds to packet loss, using fast retransmission/fast recovery instead of RTO, and then avoiding RTO's penalty.

Besides, the co-existence of incast traffic (especially short-lived incast) with elephant flows brings other troubles and challenges. Incast's flows may get queued up behind packets from large flows in presence of congestion if ever available buffer space remains, experiencing performance degradation (long queuing delay or tail drops) [9]. Switches must be able to accommodate mixed intensive communication traffic with full throughput and low latency while efficiently handling incast. Smart buffering at the switch level is then needed. It requires intelligent buffer management functions to serve mixed incast traffic and elephant flows efficiently. Cisco Nexus 9000 Series Switches propose such intelligent buffering for Data centers, using a flow classifier (Elephant Trap - ETRAP), a scheduling mechanism (Dynamic Packet Prioritization - DPP), and an active queue management scheme (Approximate Fair Dop - AFD) [10].

For this dynamic and smart buffer management to become

beneficial for incast, its performance model providing insights on how various factors affect it is needed. The literature lacks these types of models. The existing ones [5], [6], [11] are either tightly coupled with a particular protocol version or specific to certain empirical data. Motivated by this observation, we propose a machine-learning-based incast performance modeling engine capable of learning from collected historical data and predicting incast performance metrics. The learning approach has the advantage of being independent of underlying protocols and any restricted assumptions. The power of data is leveraged to achieve this prowess.

However, applying machine learning to networks controlled and managed with distributed algorithms is hard [12]. Fortunately, software-defined networking (SDN) by separating the control plane from the data plane eases control, introduces flexibility in network management, and provides a good opportunity for machine learning. Indeed, the logically centralized SDN control plane has a global network view and SDN's telemetry capabilities (INT, etc.), enabling the collection of various network data, ease the application of Machine learning approaches. We then propose the machine learning prediction approach coupled with SDN-enabled data center management.

This capability of predicting incast performance in the SDN-enabled environment may be useful for the following management tasks: smart adaptive buffer management, online global network optimization (e.g., maximization of network utilization), QoS guarantee by ensuring that performance metric and SLA are also met for performance diagnosis.

The main contributions of this paper are summarized below:

- We propose a machine learning framework build upon SDN for incast performance prediction. This service will be leveraged by smart buffering schemes and online network optimization algorithms to provide efficient performances in datacenters.
- We carry out intensive experiments with the NS-3 simulator and construct the needed dataset. Using this dataset, we designed machine learning incast completion time prediction models using random forest regression.
- And finally, we present the performance evaluation results of the machine learning models.

The rest of this paper is organized as follows. TCP incast system setup and the motivations of our work are presented in Section II. In Section III, we give a detailed presentation of our proposed framework. Section IV presents the model construction stage. An analytical model for incast performance prediction is presented in Section V. Evaluation results and analysis are provided in Section VI. We discuss related work in Section VII. Finally, we conclude this work and provide future research directions in Section VIII.

II. Setup and Motivations

A. Incast System Setup and Notations

Fig. 1 shows a simplified topology of a typical incast scenario without loss of generality. In this figure, N servers send each other the quantity SRU (Server Request Unit) simultaneously to the sink node. This corresponds to the Fixed Fragment Workload (FFW) in contrast to the Fixed Block Workload (FBW), where the total block size is fixed and partitioned amongst an increasing number of servers. We consider the setting parameters as in TABLE I. These notations hold for the rest of the paper.

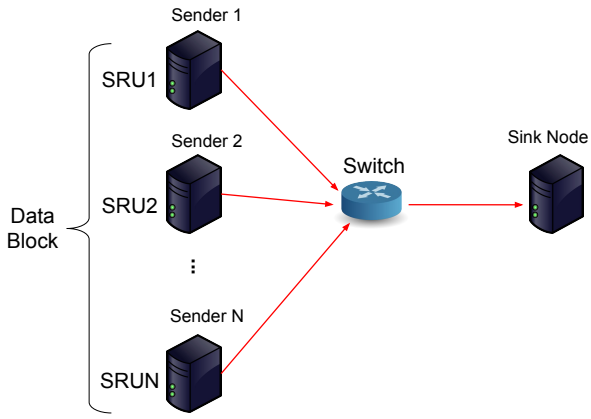

Fig. 1. Simplified topology for a typical incast scenario

TABLE I
Parameters and Notations

Parameters	Description
N	Number of competing senders
SRU	Server Request Unit size, per sender. SRU = 256 KB
B	Switch buffer size in packets. Eg. 64 pkts or 96 KB
C	Bottleneck link capacity. C = 1 Gbps
RTT_{noLoad}	RTT without queuing delay. $RTT_{noLoad} = 200us$
RTO_{min}	Minimal TCP Retransmission timeout. E.g 10 ms
S	TCP segment size, S=1446 bytes. Packet size = 1.5 KB
τ	Overall Incast Completion Time

B. Motivations

Adaptive buffer management and online network optimization are needed to achieve efficient performance for data center workloads, especially incast traffic [10], [13]. SDN, fortunately, holds one of the building blocks. Indeed it brings deep flexibility to network management, allowing the network controller to configure the network behavior up to the flow-level granularity. With its fine-grained telemetry capabilities, many possibilities for online network optimization are opened [14]. The remaining building block at this point is network performance modeling. It will allow the SDN controller to optimize Key Performance Indicators (KPIs), guarantee QoS for incast flows, and investigate different what-if scenarios and then anticipate adjustments in a proactive control and management fashions.

The classical approach for network modeling is analytic models. Analytical TCP incast performance modeling is very challenging. Indeed, TCP's stack is a complex system that involves many heuristics to handle network conditions and application behaviors [15]. Subtle changes in its parameters may

lead to completely different performance. As a consequence, one of the difficulties of analytical modeling is isolating the range of TCP variables and system variables of interest [5]. Some variables may be inter-dependent with others and more, some of them may have no impact at all on incast performance. The abundance of somewhat counter-intuitive findings from incast analytical modeling works suggests not relying solely on intuitive analysis.

The large majority, not to say all of the existing modeling work on incast, was done with the aim of analyzing the incast problem to solve it, but not with a performance prediction purpose. For example, The work in [6] focuses on how incast occurs and how various parameters affect it but not on calculating the accurate performance metric (throughput) for incast. Besides, the analytical models generally rely on observation data from simulations to support the simplification assumptions. When context changes, or with the algorithms designed to solve the incast problem, those assumptions remain difficultly valid to estimate performance metrics (throughput or completion time).

For performance prediction purposes, any proposed solutions to incast need to be evaluated under a wide variety of settings and to be modeled. This may be infeasible with only analytical approaches. A degree of autonomy could be brought using machine learning. And more there is no single solution for all scenarios. With the existence of many solutions either transport-based, application-based, or SDN-based ([4], [8], [16], [17]) to handle incast, relying only on analytical performance modeling is not a practical long-term solution. With its capability of not relying on any domain-specific assumptions, machine learning can then be leveraged to construct a generalized model via a uniform training method.

In this context, We propose a machine learning framework build upon SDN for incast performance prediction. This service will be leveraged by smart buffering schemes and online network optimization algorithms to provide efficient performances in data centers.

III. SDN-enabled Machine Learning Incast Performance Prediction Framework

By following the typical Machine learning workflow for networking as specified in [18] and leveraging SDN [19], [20], we come up with the SDN-enabled machine learning incast prediction framework in Fig. 2. Indeed, SDN is already deployed and used in data center environments [21]. The machine learning workflow for networking is very similar to the traditional machine learning one. It includes six stages as follows: Problem formulation, Data Collection, Data Analysis, Model Construction, Model Validation, and the last stage is Deployment and Inference.

The framework is based on two main cornerstones: SDN and the power of suitable machine learning algorithms of being able to learn some properties of a historical dataset and leverage the learned proprieties to provide good estimations on new observations.

From Fig. 2 the workflow of the framework is as follows. Firstly the prediction model is constructed offline by doing training and parameter tuning on the historical data. The historical dataset may be composed of a large number of samples. Each sample represents a combination of features' values and the associated target value since we are in a supervised learning configuration. The features include the congestion algorithm used (tcpCC), the queuing discipline at the switch level (qdisc), the number of competing senders (N), the bottleneck bandwidth (C), the round-trip-time (RTT), the server request unit (SRU), the minimum retransmission timeout (RTO_{min}) and the target attribute is the incast completion time (τ). Prior knowledge or "domain-specific knowledge" and insights may be leverage at this stage of model construction.

The constructed model is then deployed (1) as the Inference Agent. Care should be taken for selecting the model concerning some operational aspects such as prediction latency, stability, and accuracy of the inference, etc. The model is deployed to be used for incast performance inference. Here real-time inference is desirable. Incast may generally consist of short flows which last few microseconds. If inference on real-time input could be done in real-time too, optimization or adjustments could be done before the incast payload transfer takes place. Otherwise proactive approaches could be used.

The online input (2), composed of (tcpCC, qdisc, C, SRU, N, RTT, SRU, RTO_{min}), is got when an incast traffic is initiated by the client leveraging SDN fine-grained telemetry, In-band network telemetry (INT) and P4. Taking this input, an inference of the incast traffic's performance is done (3). This information will then be used at the control plane by smart buffering module or any traffic flow optimization algorithm to achieve efficient performance for the incast traffic and the co-existing ones. The optimization may concern, for example, network utilization maximization, global low mean delay, etc.

Finally, when the incast traffic completes, its really observed performance metric is also collected efficiently and the historical dataset could be updated (4). Having the database up-to-date is important, and will allow taking into account new dynamics from the data center. When the database significantly changed, the model needs to be re-constructed and re-deployed.

The historical data gathering and online update of the historical data with the newly collected data are crucial for our framework. The historical data could also be enriched from outside (other owned data centers, or just from the cloud). For the data gathering concern, we will take advantage of the fact that a data center operator (e.g. Amazon, Microsoft, Google, Facebook) holds diverse data centers from which data could be gathered and mutualized. Indeed, this data collection needs to be done smartly in order to have very representative data comprising the features of interest. When an incast request is initiated the corresponding bottleneck switch knows the number of servers (N) involved in the incast traffic. The available bandwidth C could be estimated with traditional monitoring tools or lightweight bandwidth estimation through low overhead byte counter collection [22]. And with INT/P4

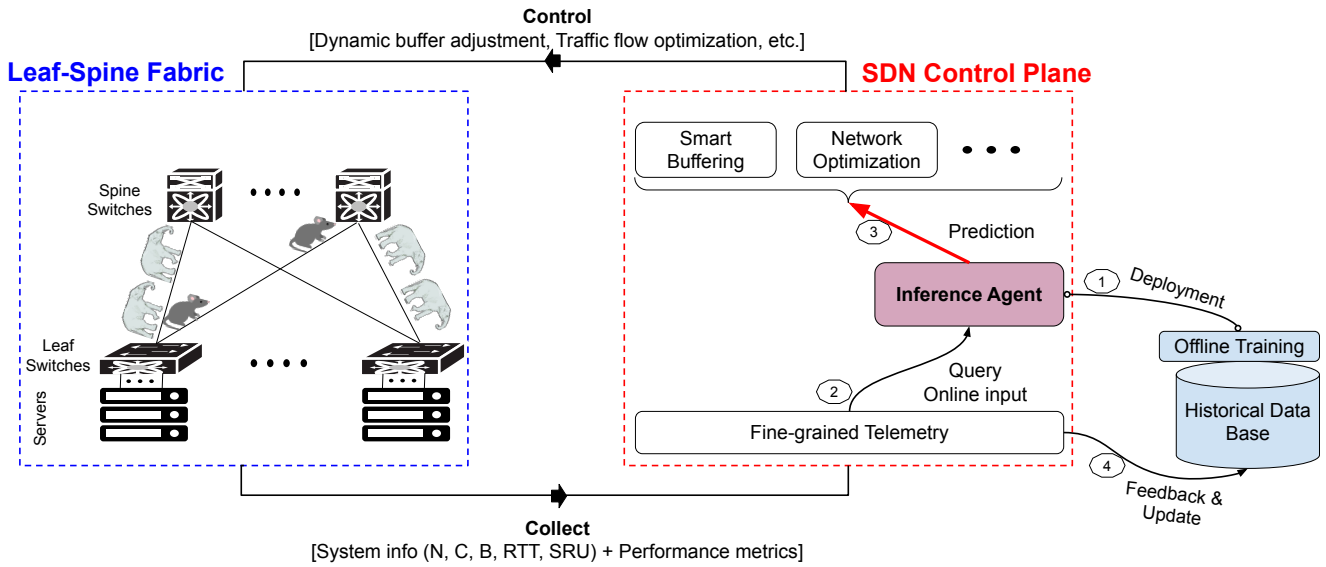

Fig. 2. SDN-enabled Learning-based Incast Performance Inference Framework

we could collect any other useful end-to-end information from the data plane.

IV. Learning-based Modeling

Recalling the workflow from [18] we begin this section with the problem formulation. For the incast performance inference, the target metric (completion time) being a continuous variable, its prediction is a regression problem. It falls under the class of supervised learning algorithms. The other main classes being unsupervised learning and reinforcement learning algorithms.

A. Dataset and Analysis

We conducted intensive NS-3 simulations using the scenario topology in Fig. 1 and varying the parameters from TABLE I. For every simulation, we compute the corresponding completion time. We finally come up with a dataset composed of 46581 observations, six parameters, and one target variable, the completion time. The variables include two categorical variables: the congestion control algorithm used (NewReno or DCTCP) and the associated queuing discipline (FIFO or FQ_CoDel for NewReno and RED-ECN for DCTCP). The numerical variables are the bottleneck link bandwidth C, the round trip time RTT, the switch buffer size B, and the number of simultaneous senders N. The server request unit SRU and RTO_{min} were respectively fixed to 256000 bytes, and 10ms and are not part of the dataset.

For the dataset preparation for training, we consider two possibilities: a single model taking six features (two categorical and four numerical variables) and the case where we consider three different training sets for the different categories (NewReno_FIFO, NewReno_FQ, and DCTCP_RED). For the individual models' case, we use then only the numerical variables as training features. TABLE II summarizes information about the different datasets used.

We then consider these two cases in the data pre-processing step. We scale our data by standardizing numerical features. It consists of centering a feature's observations to the mean and scaled it to unit variance. For the singe model, we encode the two categorical features as a one-hot numeric array. Indeed five new numerical (binary) features are created to represent the categorical features' values (NewReno, DCTCP, FIFO, FQ_CoDel, RED-ECN).

TABLE II
Datasets

Models	n_samples	n_features
Single Model	46581	6
NewReno_FIFO	15492	4
NewReno_FQ	15502	4
DCTCP_RED	15587	4

B. Model Training

After data analysis, we first investigate classical machine learning algorithms from less complex to more complex without hyper-parameter tuning in order to pick the most promising to work with. The models investigated are Linear Regression (lasso and ridge), Support Vector Regressor (SVR) with three kernels (linear, RBF, and polynomial), Decision tree, Random Forest(RF), and Multi-layer Perceptron (MLP). Random Forest only provides good results. Apart from decision trees, the other investigated machine learning algorithms were unable to capture the dataset dynamics, providing bad results. We then focus on Random Forest for the rest of this work and as a proof-of-concept implementation. The machine learning algorithms are implemented using Scikit-learn 0.23.2 [23].

Random Forest falls under machine learning averaging methods that combine the predictions of several base estimators here decision trees. The combined estimator is usually better since its variance is reduced. Decision trees are a non-parametric machine learning algorithm that predicts by learning simple decision rules inferred from the data features.

A random forest regressor has several hyper-parameters that need to be tuned for performance optimization. Some of the most important include the number of estimators (trees) used to construct the forest (n_estimators), the maximum number of features provided to each tree (max_features), max_depth which depth we want every tree in the forest to grow, etc. For example, after a certain number, increasing the number of trees has almost no accuracy improvement but just increases model complexity with high training time.

Scikit-learn provides two main tools for hyper-parameter tuning GridSearchCV and RandomizedSearchCV. GridSearchCV exhaustively considers all parameter combinations from a parameter grid. On the other hand, RandomizedSearchCV can sample a given number of candidates from a parameter space with a specified distribution, which is more convenient when we have a large search space. We use both on a set of parameter ranges, but the Scikit-learn default hyper-parameter values perform quite well. The random forest regression algorithm with 100 estimators (trees) is used for both the single model case and the individual ones.

V. Analytical Modeling

Before presenting the evaluation results of our machine-learning performance prediction approach, we present here an analytical model for predicting incast completing time when TCP NewReno is used. Timeout is the main factor of goodput degrading [11]. We use recommended small RTO_{min} in the milliseconds, which solves quite acceptably the goodput collapse, making the timeout impact almost negligible. This analytical model is compared to the machine-learning-based in the next section.

A. Assumptions

Firstly we consider that the simultaneous incast senders are fully synchronized. The overall congestion window evolution follows an aggregate AIMD. This phenomenon is called TCP Synchronization, where multiple TCP connections increase and decreasing their congestion windows simultaneously. Then all the senders will be considered as a single aggregate source sending the total data to the client.

Secondly, we consider TCP congestion steady-state. Taking a macroscopic view of the traffic sent by the aggregate source, we can ignore the slow start phase. Indeed, the connection is in the slow-start phase for a relatively short period because the connection grows out of the phase exponentially fast. When we ignore the slow-start phase, the congestion window grows linearly, gets chopped in half when loss occurs, grows linearly, gets chopped in half when loss occurs and so on.

It's worth pointing out that one RTT is required to initiate the TCP connection. After one RTT, the client sends a request for the incast data. The first bytes of the data are piggybacked onto the third segment in the three-way TCP handshake. After a total of two RTTs, the client begins to receive data from the aggregate source.

B. Modeling Completion Time of Incast

Considering the assumptions mentioned above and being inspired by [24] we propose incast completion time analytical model as follows.

Let $X = \frac{N*SRU}{S}$, the number of segments present in the incast data. Using TCP and its AIMD congestion mechanism, we have the evolution of the congestion window as follows. The first window contains 1 segment, the second window contains 2 segments, the third window contains 4 segments, and so on. More generally, the k-th window contains 2^{k-1} segments. Let K be the number of windows that cover incast data to be transmitted. K can be expressed in terms of X as follows:

$$K = min\{k : 2^0 + 2^1 + 2^2 + ... + 2^{k-1} \geq X\}$$

$$K = min\{k : 2^k - 1 \geq X\}$$

$$K = min\{k : k \geq \log_2(X + 1)\}$$

$$K = min\{k : k \geq \log_2(\frac{N * SRU}{S} + 1)\}$$

After transmitting a window's worth of data, the server may stall (i.e., stop transmitting) while it waits for an acknowledgment. But not every time. Let us now calculate the amount of stall time after transmitting the k-th window. The time from when the server begins to transmit the k-th window until when the server receives an acknowledgment for the first segment in the window is $\frac{S}{C} + RTT$. The transmission time of the k-th window is $\frac{S}{C} * 2^{k-1}$.

The stall time is the difference of these two quantities:

$$max\{(\frac{S}{C} + RTT - 2^{k-1} * \frac{S}{C}), 0\}$$

The server can potentially stall after the transmission of each of the first K-1 windows. (The server is done after the transmission of the K-th window.) We can now calculate the latency for transferring the overall incast data. The latency has three components: $2RTT$ for setting up the TCP connection and requesting incast data; $N*SRU/C$, the transmission time of the overall data; and the sum of all the stalled times. Thus, the incast completion time τ is:

$$\tau = 2*RTT + \frac{N * SRU}{C} + \sum_{k=1}^{K-1} max\{(\frac{S}{C} + RTT - 2^{k-1} * \frac{S}{C}), 0\}$$

We could obtain a more compact expression for the completion time with Equation 1 as follows:

$$\tau = 2 * RTT + \frac{N * SRU}{C} + \sum_{k=1}^{P} (\frac{S}{C} + RTT - 2^{k-1} * \frac{S}{C}) \quad (1)$$

With $P = min\{Q, K-1\}$

where $K = min\{k : k \geq \log_2(\frac{N*SRU}{S} + 1)\}$

and $Q = max\{k : k \leq \log_2(1 + \frac{C*RTT}{S}) + 1\}$

This model could be refined, approximating loss rate and including the corresponding retransmission times. However, these approximations are challenging. And with the machine learning approach, there is no need to look for such approximations. They are automatically learned from the data.

VI. Validation and Analysis

Evaluation experiments were carried out on an Intel Core i7-7500U CPU 2.70 GHz x 4 with 16 GB of RAM running Ubuntu 16.04 LTS. We consider three evaluation metrics. The first is the prediction **score** (See Eq. 2). It represents which proportion of the variance in the dependent variable is predictable from the independent variables. The most precise regression model would be the one that has a relatively high R squared, close to 1. We will represent the score in percentage. Secondly we will use **NMAE** for Normalized Mean Absolute Error (See Eq. 3). We want the NMAE to be as small as possible. And finally, we will consider the prediction time.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{\sum_{i=1}^{n}(y_i - \bar{y})} \text{ , with } \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \quad (2)$$

$$NMAE(y, \hat{y}) = \frac{\frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}}{\bar{y}} \text{ , with } \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \quad (3)$$

A. Prediction Score and Normalized Mean Absolute Error

The first presented results concern prediction accuracy represented by the prediction score and the NMAE, all in percentage. Fig. 3 shows prediction score and NMAE for different training size ratios for the single model and the individual ones. The general tendency is that the precision is quite stable with training ratios from 0.2 to 0.4, meaning a training set of 80% to 60%, respectively. More tightly, we can observe a slight decrease but not meaningful for the single model from 90.75% to 89.15%. The NMAE involves inversely with the general stability observed. The NMAE for the single model is around 20%.

The other observation is that the individual models perform better than the single model especially for NewReno_FIFO (97.78% to 97.03%) and NewReno_FQ (96.23% to 95.73%). The NMAE for NewReno_FIFO is around 7% and 8% for NewReno_FQ. However, performances are less good for DCTCP_RED where we observe scores from 83.16% to 86.21% with the NMAE around 27%. Dynamics with DCTCP is then more complex to capture, needing the investigation of other machine learning models or adding new features to improve its performance.

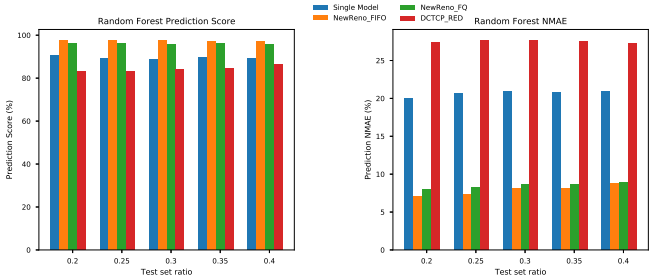

Fig. 3. Score and NMAE vs. Test Size Ratio

B. Machine learning vs. Analytical Model Predictions

Fig. 4 presents some simulation data-points from the test set, their corresponding prediction with the individual random forest model, and the prediction with the analytical model presented in Equation 1. We present the results for NewReno with both FIFO and FQ_CoDel. The machine learning prediction follows well the data-points. The analytical model even in a simple form is able to capture the data-points apart from the points where the completion time is quite high. The normalized mean absolute errors for these shown data points are presented in TABLE III (where CC stands for the congestion algorithm used and QDISC, the associated queueing discipline).

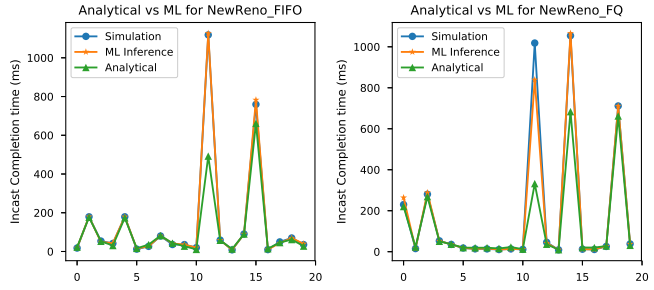

Fig. 4. Learning vs. Analytical Modeling Predictions

TABLE III
NMAE Analytical vs. ML

CC & QDISC	NewReno_FIFO	NewReno_FQ
NMAE ML	2.12 %	6.73%
NMAE Analytical	28.40%	33.22%

It's worth pointing out that the queuing discipline was not taken into account during the model construction, at least not explicitly. But supposing overall synchronization of congestion windows assumes implicitly fair-queue share and then makes the model suitable for fair queuing. Also, we note that fair queuing does not improve consequently the overall completion time. However, the bandwidth is fairly shared between senders, which is not the case with FIFO. With FIFO, some senders may finish transmitting their SRU very quickly and others too late, presenting great unfairness between senders.

C. Prediction Time Distribution

Finally we present the atomic (one-by-one) prediction latency in Fig. 5. The prediction time of the single model is slightly higher than those of the individual models since it is more complex and is constructed using all the individual training sets. However, this difference needs to be balanced with the fact that in the case of the individual models a prior process time is needed, to match input to the corresponding model.

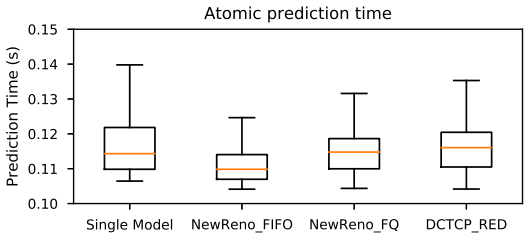

Fig. 5. Atomic Runtime Prediction Latency Distribution

The prediction time is unfortunately high, around 0.10 seconds. This high latency can be explained by the use of Scikit-learn. Indeed, Scikit-learn is not necessarily suitable for production model deployment but more suitable for prototyping. Scikit-learn implements some methods in C for performance improvements but additional overhead is present due to Python function calls, feature extraction, to name a few. Suitable input data format usage could improve performance and also bulk prediction (many instances at the same time). Indeed when predicting bulk test sets, we have quite the same prediction latency as for atomic prediction and then the prediction throughput increases. The performance gain with bulk prediction can be explained with these factors: linear algebra libraries optimizations, branching predictability, CPU cache, etc. It's also pointing out that existing optimization solutions for machine learning pipelines are generally dedicated to the training step.

In production, more optimized implementations and frameworks coupled with specialized hardware (FPGA, GPU, TPU, Xilinx, etc.) are needed [25]. These hardware accelerators include FPGA (used, for example, by Microsoft and Xilinx ML Suites), Nvidia's GPU, AI ASICs (e.g. Google's TPU), etc. For our proposed inference system, we hope there will be enough computation resources in data center management and control planes, and the use of dedicated hardware will improve its performance. This way advantages of this machine learning inference could be effectively beneficial for overall flow QoS optimization in data centers.

Moreover, the use of the proactive approach where the control plane simulates what-if-scenarios by exploring some incast setups and parameter adjustments taking the prediction of the inference agent can help. Parameter adjustments needed to achieve global performance can then be anticipated. In this case, the prediction time penalty will be less severe.

VII. Related Works

A. TCP Incast Modeling

The authors in [5] analyze the dynamics of the incast problem by exploring its sensitivity to various system parameters. The understanding of the dynamics of incast is done with an analytical model based on empirical data. This quantitative model is completed with a qualitative refinement to capture most of the incast aspects, unfortunately, not all. This work, however, explains the root cause of incast, the RTO, and supports the TCP-level solution consisting mostly of using small RTO_{min} values in the data center RTT scales to alleviate throughput collapse.

The work [11] provides an analytical goodput model of incast where the goodput deterioration is explained by 2 types of timeouts. The block-tail timeout is observed when the number of simultaneous senders N is small, and the block-head timeout when N is large. This work considers the sending of consecutive data blocks. The analytical model characterizes well the general tendency of the TCP incast problem. This helps to understand the problem and helps understanding possible solutions such as RTO_{min} reduction. But for a new solution to handle incast traffic, we may need to rebuild a new model to express attended performances.

Finally, authors in [6] provide an in-depth understanding of how TCP incast problem happens with an interpretive model. This model explains qualitatively how systems parameters (block size, link capacity, buffer size) and mechanism variables (RTO_{min}) impact TCP incast.

B. Machine Learning for QoE / QoS inference in SDN

A comprehensive survey on machine learning algorithms' application to SDN can be found in [12]. This application to SDN is guided by diverse objectives that include traffic classification, security, resource management, routing optimization, and finally Quality of service (QoS) / Quality of Experience (QoE) prediction. For this latter let us focus on two works [26] and [27].

An end-to-end application QoS prediction is proposed in [26]. OpenFlow per port statistics are used to infer the service-level QoS metrics such as frame rate or response time for video-on-demand applications. Two machine learning algorithms (decision tree and random forest) are used.

The authors in [27] propose a two-phase analysis approach for QoS inference, able to predict traffic congestion. Firstly it discovers which key performance indicators (KPIs) are correlated with the QoS metric using a decision tree. Then it mines each KPI's quantitative impact using linear regression.

The work in [28] proposes RouteNet that leverages the ability of Graph Neural Networks (GNN) for network modeling and optimization in SDN. Taking as input network topology information, routing schemes, and traffic matrix RouteNet, based on Generalized Linear Models, can provide accurate source-destination KPIs such as delay distribution (mean delay and jitter) and packet drop prediction. These KPI predictions could be leveraged by a QoS-aware optimizer to improve global performance.

VIII. Conclusion And Future Research Directions

In this work, we propose an SDN-enabled machine learning incast performance prediction framework for data center networks. This framework's goal is to provide incast completion time inference at run-time. This information could then be leveraged by any flow optimization algorithm or adaptive smart buffering mechanism to dynamically adjust system parameters to achieve efficient performance for both incast traffic and the co-existing traffic (generally elephant flows). We conduct intensive NS-3 simulations and construct a representative dataset. After that, a random forest regression model was implemented.

The evaluation results show that the proposed learning-based incast performance inference can provide good predictions either using a single model or individual models depending on the congestion control algorithm and queuing discipline used. We achieve up to 90% of prediction performance score for the single model case, 97% for TCP New Reno with FIFO, 97% for NewReno with FQ_CoDel, and 86% for DCTCP with RED and ECN. We also compared our random forest model to the analytical model approach. The machine learning approach has the advantage of being easily generalizable for diverse congestion control and queuing discipline schemes, dynamic environment, and of not being tightly coupled with any domain-specific assumptions and approximations.

As future works, we plan to implement a neural network inference model and investigate prediction latency optimization solutions. We will also extend the incast traffic dataset with new congestion control and queuing discipline schemes (e.g BBR) and by integrating new features in the dataset as the SRU, RTO_{min}, etc.

References

[1] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 29–42.

[2] Y. Zhang and N. Ansari, "On architecture design, congestion notification, tcp incast and power consumption in data centers," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 39–64, 2012.

[3] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems." in *FAST*, vol. 8, 2008, pp. 1–14.

[4] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," *ACM SIGCOMM computer communication review*, vol. 39, no. 4, pp. 303–314, 2009.

[5] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009, pp. 73–82.

[6] W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, and F. Baker, "Comprehensive understanding of tcp incast problem," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1688–1696.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 63–74.

[8] Y. Xu, S. Shukla, Z. Guo, S. Liu, A. S.-W. Tam, K. Xi, and H. J. Chao, "Rapid: Avoiding tcp incast throughput collapse in public clouds with intelligent packet discarding," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1911–1923, 2019.

[9] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[10] "Intelligent buffer management on cisco nexus 9000 series switches," https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.pdf, 2017.

[11] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding tcp incast in data center networks," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 1377–1385.

[12] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

[13] P. Chuprikov, S. Nikolenko, and K. Kogan, "Towards declarative self-adapting buffer management," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 3, pp. 30–37, 2020.

[14] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," in *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, pp. 140–151.

[15] Y. Li, R. Miao, M. Alizadeh, and M. Yu, "{DETER}: Deterministic {TCP} replay for performance diagnosis," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 437–452.

[16] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Dctcp: Efficient packet transport for the commoditized data center," 2010.

[17] A. M. Abdelmoniem, B. Bensaou, and A. J. Abu, "Sicc: Sdn-based incast congestion control for data centers," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[18] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, 2017.

[19] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[20] N. Foster, N. McKeown, J. Rexford, G. Parulkar, L. Peterson, and O. Sunay, "Using deep programmability to put network owners in control," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 4, pp. 82–88, 2020.

[21] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[22] K. B. Nougnanke, M. Bruyère, and Y. Labit, "Low-overhead near-real-time flow statistics collection in sdn," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 155–159.

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

[24] "Modeling latency: Dynamic congestion window," http://www2.ic.uff.br/~michael/kr1999/3-transport/3_07-congestion.html.

[25] D. Crankshaw, "The design and implementation of low-latency prediction serving systems," Ph.D. dissertation, UC Berkeley, 2019.

[26] R. Pasquini and R. Stadler, "Learning end-to-end application qos from openflow switch statistics," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–9.

[27] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate, "Applying big data technologies to manage qos in an sdn," in *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE, 2016, pp. 302–306.

[28] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet: Leveraging graph neural networks for network modeling and optimization in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, 2020.