

Auric: Using Data-driven Recommendation to Automatically Generate Cellular Configuration

Ajay Mahimkar, Ashiwan Sivakumar, Zihui Ge, Shomik Pathak, Karunasish Biswas

AT&T, USA

{mahimkar,ashiwan.sivakumar,gezihui,shomik.pathak,karunasish.biswas}@att.com

ABSTRACT

Cellular service providers add carriers in the network in order to support the increasing demand in voice and data traffic and provide good quality of service to the users. Addition of new carriers requires the network operators to accurately configure their parameters for the desired behaviors. This is a challenging problem because of the large number of parameters related to various functions like user mobility, interference management and load balancing. Furthermore, the same parameters can have varying values across different locations to manage user and traffic behaviors as planned and respond appropriately to different signal propagation patterns and interference. Manual configuration is time-consuming, tedious and error-prone, which could result in poor quality of service. In this paper, we propose a new data-driven recommendation approach **Auric** to automatically and accurately generate configuration parameters for new carriers added in cellular networks. Our approach incorporates new algorithms based on collaborative filtering and geographical proximity to automatically determine similarity across existing carriers. We conduct a thorough evaluation using real-world LTE network data and observe a high accuracy (96%) across a large number of carriers and configuration parameters. We also share experiences from our deployment and use of Auric in production environments.

CCS CONCEPTS

• **Networks** → **Network management; Wireless access points, base stations and infrastructure; Network experimentation;** • **Computing methodologies** → **Machine learning approaches.**

KEYWORDS

Cellular network configuration, carrier addition, collaborative filtering, recommendation algorithms

ACM Reference Format:

Ajay Mahimkar, Ashiwan Sivakumar, Zihui Ge, Shomik Pathak, Karunasish Biswas. 2021. Auric: Using Data-driven Recommendation to Automatically Generate Cellular Configuration. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21)*, August 23–28, 2021, Virtual Event, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3452296.3472906>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8383-7/21/08...\$15.00

<https://doi.org/10.1145/3452296.3472906>

Ethical issues: This work does not raise any ethical issues.

1 INTRODUCTION

In recent years, we have seen a massive explosion in applications relying on cellular service ranging from mission critical applications such as first responder services, medical emergencies, police communications, to business transactions such as stock trading as well as user entertainment such as high-definition video, online gaming, and virtual reality. These applications have introduced significant increase in voice, video, and data traffic to be carried over the cellular network thereby placing tremendous requirements on the network capacity. We have seen a rapid evolution in the cellular service technologies from UMTS to LTE, and recently 5G. Cellular service providers continuously add carriers (also, referred to as radio channels) in their networks in order to keep up with the increasing demand in traffic and provide high quality of experience to the users.

Using real-world network data collected over three years from a large LTE service provider in the US, we observe that there is a tremendous increase in traffic, and numbers of carriers. Carrier addition in the LTE network often involves physical hardware installation to create the new radio channels and then software configuration to integrate into the network. It is thus an expensive process and one must carefully plan where to add new carriers based on the capacity and coverage needs. After carriers are added to the network, it is important to accurately configure them to efficiently carry the user traffic and offer the best quality of experience to the users. Any mis-configuration or sub-optimal configuration increases the risk of a poor quality of experience, thereby rendering the carrier addition a waste of money and effort. There are on the order of thousands of configuration parameters related to user mobility, interference, load balancing, handovers, outage and congestion management that make the configuration effort extremely challenging. Furthermore, different geographical locations exhibit different signal propagation patterns, and thus there is a need to tune the configuration according to the location characteristics (e.g., downtown area versus national park). The dependencies between the various parameters and the possible ranges within which they can be set make the configuration management even more challenging.

State of the art and limitations. The standard practice in cellular configuration management is the creation of the **rule-books guided by domain knowledge** that define the configuration parameter values across different attributes of carriers. We define an **attribute** of a carrier using its characteristics such as carrier frequency, type, size, or downlink channel bandwidth. The rule-books are created by the domain experts (network engineers) and sometimes with help from radio equipment vendors. They are often

referenced to, when generating the configuration parameter values for the newly added carrier. Capturing the domain knowledge accurately and maintaining it is quite challenging, time-consuming, tedious and sometimes error-prone. This is because of the large number of configuration parameters, complex interactions between parameters, impact of changes on service performance, and continuous evolution of networks through software upgrades and technology updates. Recently, automated solutions based on self organizing networks (SON) [2, 3, 33] are being used in operational networks to safely implement the configuration on the carriers based on the rule-books. However, even with SON, the rule-books are created and maintained using domain knowledge. When new carriers are integrated, SON only ensures the configured values are compliant with the rulebook. Further, SON is still unable to determine an appropriate value in case a parameter has a range to choose from. Updates can happen frequently to the rule-books, and thus management of these rule-books and configuration parameters across the network is a continuous process. SON also configures parameters such as the carrier identity and immediate neighbor lists that are specific to individual carriers and cannot be repeated anywhere in the network. Such parameters are either designed before the launch process or measured directly after the actual deployment, and is outside the scope of our stated problem and solution.

The parameters that do take a wide range of values are tuned by the network engineers differently across different locations. The reason behind this diverse configuration setting across different locations is to carefully account for the changing traffic distribution, user mobility patterns, signal propagation, and radio channel conditions. This makes it quite hard to capture the values for the configuration parameters using well-defined rules. Such parameters are then typically configured manually by the network engineering teams based on their operational experiences, trial-and-observe methods, and best practices. This incurs time, cost, and effort. An example parameter is a capacity threshold to control load balancing actions for distributing traffic across carriers. It takes values between 0 and 100 and can be tuned differently across different locations. This presents a unique opportunity in generating the carrier configuration for new carriers as they launch across different parts of the network and is the focus of our work.

Our approach. We propose a novel data-driven approach Auric for automatic and accurate generation of new carrier configuration parameter values using learning and recommendation techniques. Instead of having domain experts define and maintain the rule-books to manage cellular network configuration, our idea in Auric is to automatically learn the rules based on existing carrier configurations in operational networks. The intuition is carriers share similarity in their configuration across specific locations and for matching sets of attributes. For example, carriers serving downtown locations in densely populated urban areas have similar configurations. Thus, if we can identify carriers matching to the newly added one, we can leverage the configuration from matching carriers and copy/paste them to the new carrier. The challenge is how to systematically deal with diverse configuration settings across different locations. We address this by using **collaborative filtering** and **geographical proximity** to automatically identify parameter similarity across multiple carriers. Our idea is akin to algorithms used in popular

content recommendation system for new users [8, 25, 34, 36]. We believe, we are the **first** to apply collaborative filtering based recommendation algorithms for carrier configuration generation in cellular networks.

Our contributions.

1. We present the design and implementation of Auric using learning algorithms and geographical proximity to generate the configuration for newly added carriers in cellular networks.
2. We conduct thorough evaluation using real-world configurations collected from existing operational LTE cellular network across 400K+ carriers and demonstrate that Auric achieves an accuracy of 96%. For the 4% recommendations from Auric that did not match the current carrier configurations, we validated them through engineering team feedback, and identified that 28% of those mismatches turned out to be better configurations. As an added bonus, we implemented them as configuration changes (15K+ across multiple carriers and parameters) to align with the engineer expectations.
3. We share our experiences from the production deployment and use of Auric over two months in operational LTE cellular networks to generating configurations for newly added carriers.

2 BACKGROUND AND MOTIVATION

In this section, we first present background on LTE network, carrier additions and managing configuration. We then present operational practices to manage carrier configuration followed by related work. Finally, we conduct analysis using real-world data collected from a large operational cellular service and highlight the challenges in accurately determining the carrier configuration.

2.1 LTE network

Fig.1 shows the basic architecture for the LTE network that consists of the radio access network (RAN), and the core network. The RAN primarily consists of base stations also referred to as enhanced NodeB (eNodeB). The primary function of the eNodeB include radio resource management, admission control, mobility management, and dynamic scheduling. The core network consists of service gateway (S-GW), packet data network gateway (P-GW), mobility management entity (MME), policy control and charging rules function (PCRF), and home subscriber server (HSS). S-GW handles user IP packet routing and forwarding, as well as accounting for users. P-GW is responsible for IP address allocation for users, quality of service enforcement and flow-based charging according to the rules from PCRF. P-GW communicates with the Internet and voice IMS (Internet Multimedia Subsystem) core. MME handles control plane traffic and is responsible for signaling, user authentication and roaming functions. HSS manages user subscription.

eNodeB has three logical interfaces. S1-U carries user plane traffic between eNodeB and S-GW. S1-MME carries control plane traffic between eNodeB and MME. Between two eNodeBs, the X2 interface carries control plane traffic (for user mobility such as inter-eNodeB handovers) as well as data plane traffic (data forwarding between source and target eNodeB to support lossless handovers). All S1 and X2 interfaces share the same IP-based transport network. eNodeB typically divides the 360 degree coverage into 3 faces with each face consisting of multiple carriers. A carrier is a radio channel on

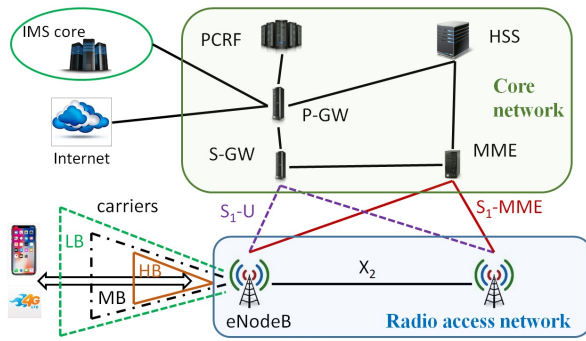


Figure 1: Logical LTE network architecture.

the eNodeB used to carry the user data and signaling messages. In LTE network, the carrier is also the center frequency of the spectrum used to transmit the LTE signal. The users connect to the eNodeB on specific carriers. Each of the carriers operates on a specific frequency band (e.g., 700 MHz) and channel bandwidth (e.g., 20 MHz). The power set, the antenna type and tilting help define coverage area of the carrier.

Multiple carriers within the face operate on different frequency bands - high band (HB), middle band (MB) and low band (LB). Due to broader reach of the LB carriers and lower interference in MB/HB, the service providers configure eNodeBs to direct the users to connect first to high bands and after the high bands get congested or run out of coverage, connection to the mid and low bands are allowed. The function of managing users across frequency bands is also referred to as *carrier layer management*. The service providers add more carriers to the eNodeB to increase either the coverage or the capacity and support growth in user traffic. It is then important to accurately configure the newly added carriers in order to ensure the carrier layer management functionality operates as desired.

2.2 Managing configuration

The configuration of the carriers in the LTE network can be categorized into multiple functions such as radio connection management, power control, link adaptation, scheduling, capacity management, layer management, and user mobility management. We describe a few configuration parameters:

1. **actInterFreqLB** - activates the inter-carrier frequency load balancing (IFLB) feature. If IFLB is activated ("true"), inter-frequency load measurements are performed by the eNodeB per carrier. Depending on the load of the users across carriers, the eNodeB can choose to hand over the users to under-utilized overlapping or neighboring carrier cells on different carrier frequencies.
2. **sFreqPrio** - enables comparison and prioritization between two candidate carriers. This comparison is done using the average load in the uplink. This parameter can help biased selection towards a carrier by giving it a higher priority. sFreqPrio can range between 1 and 10000. 1 (default) represents the highest priority and 10000 represents the lowest.
3. **hysA3Offset** - enables selection of handover margins for better handovers across carriers on the same frequency. It can range between 0 and 15 in step sizes of 0.5.

4. **pMax** - defines the maximum output power of the carrier in dBm. The maximum output power is the maximum value of the linear sum of the power across all downlink resources that is allowed to be used in a cell. The range for pMax is between 0 and 60 in step size of 0.6.
5. **qrxlevmin** - specifies the minimum required receive RSRP (reference signal receive power) level in the carrier. It ranges from -156 to -44.
6. **inactivityTimer** - captures the time period for the indication of the user inactivity in both downlink and uplink directions. The range for inactivityTimer is 1 to 65535.

There are on the order of thousands of such configuration parameters across different functions. The modeling and naming convention of configuration parameters differs across different vendors because of the lack of standardization. We formulate the configuration recommendation problem independently for each vendor and across multiple carriers in the cellular network.

Table 1: Carrier attributes.

Carrier attribute	Type	Example values
Carrier frequency	Static	700 MHz, 1900 MHz
Carrier type	Static	FirstNet, NB-IoT
Carrier information	Static	5G colocated, border
Morphology	Static	Urban, suburban, rural
Channel bandwidth	Static	10 MHz, 15 MHz, 20 MHz
Downlink MIMO mode	Static	Closed loop MIMO, 4x4
Hardware configuration	Static	RRH1, RRH2 (remote radio head)
Expected cell size	Static	2 miles, 3 miles
Tracking area code	Static	8888, 9999
Market	Static	New York, California, Texas
Vendor	Static	VendorA, VendorB, VendorC
Neighbor channel	Static	444, 555, 666
Neighbors on same eNodeB	Dynamic	8,9,10
Software version	Dynamic	RAN20Q1, RAN20Q2

2.3 Carrier attribute

A carrier attribute represents the description of the carrier such as carrier frequency, type (e.g., serving first responder network - FirstNet [4] or narrowband IoT [5]- Internet of things), morphology (urban, rural, or suburban), software or hardware version, expected cell size, market, vendor, or channel bandwidth. We describe our carrier attribute set in Table 1. As you can see, some of these attributes are static (*i.e.*, do not change for the carrier over time) such as carrier frequency, morphology, or channel bandwidth, and others are dynamic (*i.e.*, can slowly change over time) such as software version, or the number of neighboring carriers on the same eNodeB. In the next section, we describe the operational practices today for carrier configuration management and related work.

2.4 Operational practices

Given the large number of carriers and their configuration parameters in LTE service provider networks, it becomes a challenging problem to effectively manage carrier configurations. The configuration parameters can take different values across different locations and have to be continuously monitored and managed by the network operations teams. Dependencies across configuration parameters further increase the complexity. Today's operational practice includes creating rule-books that define the default parameter values for different attributes of carriers. The rule-books are created by the network engineers who possess the domain expertise and experience in LTE cellular networks, and sometimes

also with help from eNodeB vendors. The rule-books undergo periodic changes when incorporating new service features, optimizing configuration for improving service quality, or introducing new technologies (e.g., 5G). The rule-books are then used by the network engineers across different parts of the network to ensure there is consistency between the configuration in the production setup and expectation defined within the rule-book. Recently SON solutions (Self Organizing Networks [1–3, 33]) have been deployed to automatically implement the configuration changes but they still are driven by a rules-based approach.

Some of the configuration parameters (like `actInterFreqLB`) take only a few values (enumeration), versus others take values within a large range (like `sFreqPrio`). For the range parameters, the rule-books define the initial default value that is typically used. However, in several parts of the network, the engineers can choose to continuously tune and modify these range parameters to fit the needs of that area and improve service quality of experience. They ensure that the parameter values stay within the range and observe the performance impact of the parameter change to decide if they would like to keep the change or roll it back. We will show in Section 2.6 that operational cellular networks do have high variability for the range parameters. The automated configuration solutions in SON can only assign the default value and cannot replicate human intuition to be able to assign from a range. They can verify that the parameters conform to the ranges but cannot automatically discover what the optimized values are in the network. When new carriers are added to the eNodeB, the engineers have to ensure that their configurations are appropriately set based on past intuitions and optimizations.

2.5 Related work

We will present related work in the area of general network configuration management and explore if and how those techniques can be applied for LTE carriers.

Configuration synthesis. The focus is to automatically derive the low level node-specific configuration based on the high-level intent. AED [7] proposes to incrementally synthesize router configuration based on high-level policies and uses a SMT-based (Satisfiability Modulo Theory) constraint optimization formulation. Net-Complete [13] helps with updating existing network-wide configurations based on new routing policies. It auto-completes the input configurations with holes based on target values. Config2Spec [12] automatically synthesizes a formal specification of network policies based on the configuration and failure models. Propane [10, 11] focuses on synthesizing provably-correct BGP (border gateway protocol) configurations using high-level specifications of topology, routing policy, and fault-tolerance requirements. PGA [32] uses high-level policy graph abstraction (PGA) to enable simple expressions of network policies and efficient detection and resolution of policy conflicts. NetGen [35] presents a new language that allows the user to express desired re-routings and identify minimal changes to the network that satisfies the policy. Robotron [39] from Facebook is a system for managing massive production network in a top-down fashion and enabling operators to express high-level design intent and translate that automatically into low-level device configurations for safe deployment across the network. Jinjing [41]

from Alibaba enables network operators to automatically and correctly update ACL configurations in wide area networks using a set of novel verification and synthesis techniques to rigorously guarantee the correctness of update plans. NetCraft [26] from Alibaba proposes a new framework to automate the life cycle management of network configurations using a unified network model. [40] from Facebook enables holistic network configuration from design, testing, co-ordination, deployment and verification. CPR [16] automatically computes correct, minimal fixes to the network control planes by formulating the problem using MaxSMT and capturing the dependencies between control plane and different traffic classes.

Configuration verification. The focus is to verify that the current configuration in the network matches the expectation and guarantees different properties of the network such as reachability, load balancing, and service performance. Minesweeper [9] translates network configuration into logical formulae to capture the stable state of the network forwarding and accounting for interactions across multiple routing protocols such as OSPF and BGP. ARC [17] uses network configuration to verify invariants under different failure scenarios and without the need to generate the data plane traffic. Anteater [30] checks invariants defined for the data planes by translating the high-level invariants into Boolean Satisfiability (SAT) problems. HSA [24] (header space analysis) performs static analysis of the protocol headers to identify failure classes related to reachability, packet forwarding, and traffic loss. Batfish [15] detects configuration errors before configuration is applied by conducting a what-if analysis. ERA [14] focuses on network availability, reachability, and security by efficient reasoning of intended network policies. Plankton [31] scales up the model-checking approach for verification by combining with equivalence based partitioning and using optimizations such as state hashing, partial order reduction and policy based pruning. GRoot [22] uses formal semantic model and equivalence classes to verify DNS configurations. NetDice [37] proposes a scalable approach for probabilistic verification of network properties for BGP and IGP. SELFSTARTER [23] focuses on identifying network misconfigurations through automatic template inference and then detecting the deviations in configuration compared to the templates. ctests [38] propose testing configuration changes in the context of code that would be affected by the changes. Violet [21] detects specious configuration (negative performance impact of configuration changes) by using new selective symbolic execution techniques to systematically reason about the performance effect of configuration parameters. Mercury [29], PRISM [27], Litmus [28] propose statistical analysis techniques to do a before and after comparison of time-series around configuration changes and detect negative impacts to service performance. [19] from Google automate deployment of changes consistently across control as well as data planes and monitors network operational invariants.

Remarks. Since our goal is to generate configuration, we fall under the category of synthesis. The high-level intent in all of the above related work is input by the user (domain expert) and can be viewed analogous to creating the rule-books as described in the operational practices (Section 2.4). The low-level node-specific configuration for carrier would be the specific configuration commands or APIs provided by the vendors. None of the synthesis

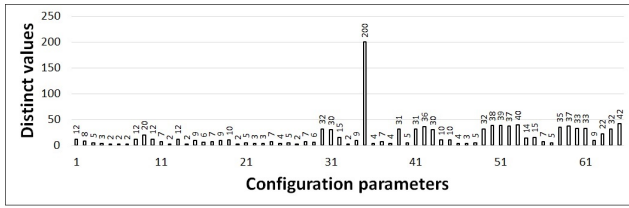


Figure 2: Distinct values across configuration.

works look to automatically derive the intent with the exception of Config2Spec [12] and AED [7]. For AED and Config2Spec, the assumption is the intent or configuration is similar across nodes with the same roles (e.g., routers in the access network should have similar configuration albeit different than other roles such as in the core network). Given the role, one can mine the configuration intent. However, in our case, it is not easy to slice and dice the carriers across different roles and thus our problem is to discover clusters of carriers with common roles or objectives. The discovery or learning part has resemblance to verification systems that aim to build models of normal behaviors shared across different roles. Several verification systems are rules-based and thus are analogous to the rule-book approach. One exception is SELFSTARTER [23] - it comes closest to our work for automated template inference. But again similar to AED and Config2Spec, SELFSTARTER needs input about the role of the node to identify the templates. Thus, our goal of not only detecting the role, but also the template is a unique problem not handled by any of the existing works. We further take a data-driven approach to discover matching nodes (or, carriers) based on attributes and recommend configuration using the patterns across the matching nodes.

2.6 Data analysis

We collect and analyze configuration parameter data collected from carriers in a large LTE network in order to show the importance and the challenging nature of the problem. We first carefully eliminate parameters that have values specific to individual carriers (e.g., IP address, carrier ID). Next, we analyze 3000+ parameters on each of the approximately 400K+ carriers and identify 65 parameters that take values within a range. The network engineering teams also confirmed with us that they often tune these 65 configuration parameters to improve service performance. The remainder parameters were taking values from an enumeration and thus could be represented using existing rulebooks. Thus, our focus for the rest of the paper will be on these 65 range parameters that take different values across different carriers.

We define the variability of a configuration parameter using the number of distinct values that it takes. Fig. 2 shows the number of distinct values for all of the 65 configuration parameters across the network. Several configuration parameters have the number of distinct values greater than 10. One of the configuration parameters has a variability of 200. Generating the value for this parameter for a newly added carrier can take any of the 200 values.

Next, we divide the 400K+ carriers into 28 markets. A market is defined as a collection of carriers that are typically managed by a group of engineers. Think of a market analogous to a state in the US. Our thesis behind the division into 28 markets was to study if

the variability is more prevalent in specific markets or is uniformly spread across them. Fig. 3 shows the distinct values across 65 configuration parameters and for each of the 28 markets. We observe that the variability (or, the number of distinct values) is quite high for some markets and for some collection of configuration parameters. Additionally, we calculate the skewness for a configuration parameter using the distribution of its values across 28 markets. Skewness [6] captures a measure of asymmetry of the distribution of the configuration parameter values around its mean and is calculated using $\frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^3}{(\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2)^{\frac{3}{2}}}$. If the skewness is between -0.5 and 0.5, the distribution is approximately symmetric. If it is less than -1 or greater than 1, the distribution is highly skewed. If it is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed. We plot the skewness across 28 markets for all 65 parameters in Fig. 4 and observe that 33 out of the 65 parameters are highly skewed and 12 are moderately skewed.

This simple analysis helped us confirm that the configuration has a high degree of variability and skewness across the carriers and markets. Thus, generating the configuration parameter values for the group of highly variable and skewed set of parameters becomes a challenge, especially when trying to derive it manually or using a rule-book.

3 AURIC DESIGN AND IMPLEMENTATION

In this section, we present the design and implementation of Auric. Fig. 5 presents an overview of Auric. Our key idea in Auric is akin to collaborative filtering for new users (often referred to as cold start [8, 25, 34, 36] problem) in content recommendation systems. Since the new carriers are not yet carrying user traffic, they are analogous to new users without any past interactions with items. Thus, we rely on the attributes of the new carriers that are available when they are activated but not carrying user traffic. We use the intuition that the configuration parameters on the new carrier are highly similar to existing carriers that have matching attribute values. As shown in Fig. 5, the carrier attributes range from carrier frequency, type, downlink channel bandwidth, morphology, cell size, market and hardware and software version. In Auric, we first learn the dependency models between the configuration parameters and the attributes for existing carriers that are carrying live traffic. Then, we recommend or predict the configuration parameter values for the new carriers using the new carrier’s attributes and the previously learned dependency models.

3.1 Problem formulation

We now formulate the problem to be solved in Auric. First, we set the notations as shown in Table 2. Fig. 6 shows the matrices for attributes and configuration parameters of the carriers. The carrier attribute matrix X serves as the predictor and the configuration parameter vector Y serves as the predictee. We build a dependency model $Y^{(i)} = \beta^{(i)} \cdot X$ for each parameter vector Y^i using attributes and configuration parameter values for the existing carriers (e.g., C_1, C_2, \dots, C_{N-1}). Then, for a new carrier C_N , we can compute the prediction of the configuration parameter $Y_N^{(i)} = \beta^{(i)} \cdot X_{N,*}$. Given this formulation, we can explore multiple supervised techniques for dependency model learning and recommendation or prediction.

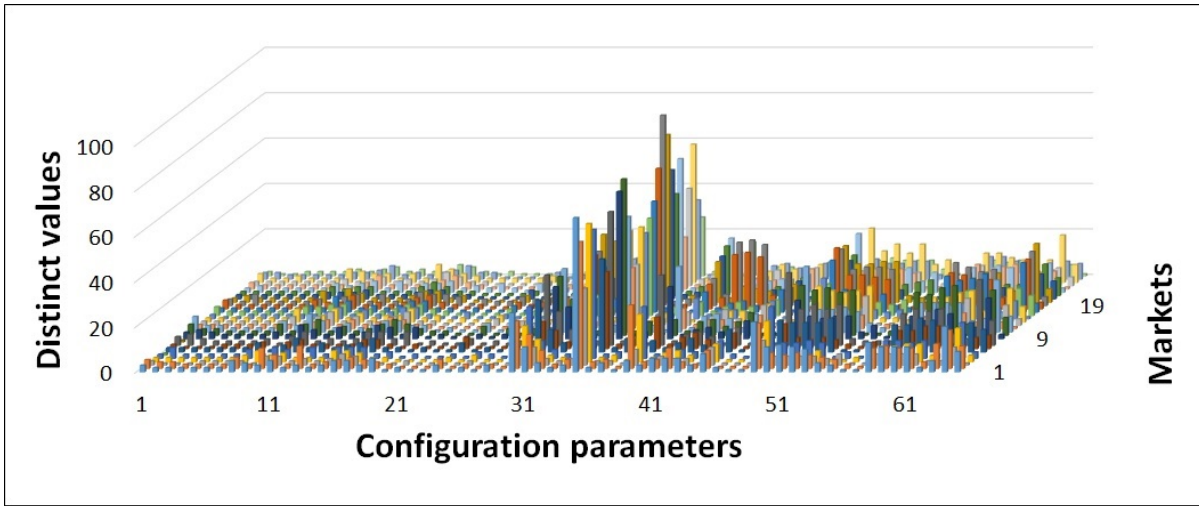


Figure 3: Distinct values across configuration parameters for each market.

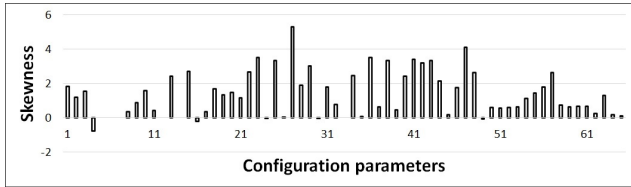


Figure 4: Skewness of configuration parameter values across markets.

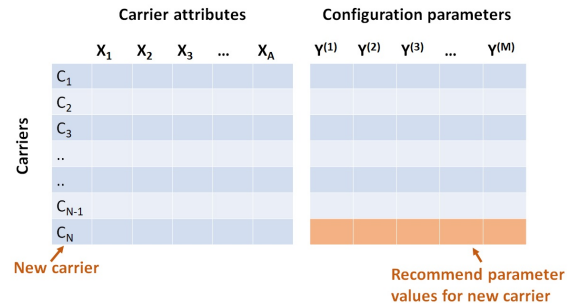


Figure 6: N carriers, A attributes for attribute matrix X . Y is matrix of M parameters for N carriers.

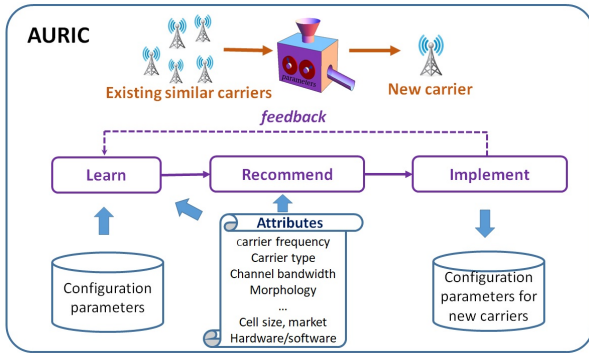


Figure 5: Auric overview.

Table 2: Notations used in Auric.

N	Total number of carriers in the network ($j = 1..N$)
M	Total number of configuration parameters for the carriers ($i = 1..M$)
C_j	j^{th} carrier
$Y^{(i)}$	vector of size N for the i^{th} configuration parameter
$Y_j^{(i)}$	value for the i^{th} configuration parameter and j^{th} carrier
$Y_{j,k}^{(i)}$	value for the i^{th} parameter and between j^{th} carrier and its neighbor k
A	Total number of carrier attributes
X	matrix of size $N \times A$ for A attributes across N carriers
$X_{j,a}$	attribute vector of size A for the j^{th} carrier

Since X and Y can contain nominal variables, we use one-hot encoding to translate them. For example, if a hardware version attribute takes values H_1, H_2, H_3 , we create three columns in X for H_1, H_2 and H_3 and the corresponding value for the carrier is binary (0 or 1).

3.2 Dependency learning

One can employ linear or non-linear regression models to discover the dependency structure or the beta coefficients ($\beta^{(i)}$). For linear regression, one can add regularization function to increase the sparsity in the solution. Typically, the configuration parameter values should be associated with a small number of carrier attributes and thus the regularization function plays a key role in discovering sparse dependency models. Lasso regression is expressed as

$$\text{minimize } \|Y^{(i)} - \beta^{(i)} \cdot X\|_2 + \lambda \|\beta^{(i)}\|_1 \quad (1)$$

The regularization parameter is $\lambda \in [0, 1]$ and l_p norm is

$$\|\beta^{(i)}\|_p = (|\beta_1^{(i)}|^p + |\beta_2^{(i)}|^p + \dots + |\beta_A^{(i)}|^p)^{\frac{1}{p}} \quad (2)$$

For non-linear regression, we bring in deep neural networks [18] (also, known as multi-layer perceptrons) that consists of multiple hidden layers (or, neurons) between the predictor carrier attributes (or, input layer X) and the predictee configuration parameters (or, output variable Y). An example 3-hidden layer fully connected deep neural network is shown in Fig. 7. Compared to linear regression models, deep neural networks can discover complex dependencies between parameters and attributes. One well-known challenge with deep neural network is the explanation for the prediction. k -nearest neighbors use distance between attributes to find the

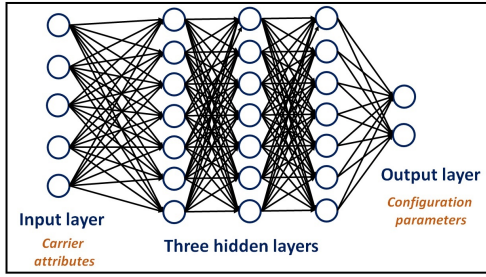


Figure 7: A deep neural network between input layer of carrier attributes, output layer of configuration parameter and three hidden layers.

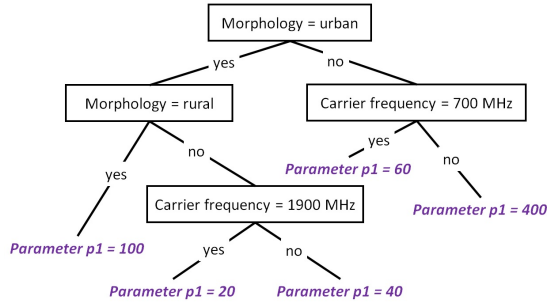


Figure 8: An example decision tree learner that presents a simple explanation for the configuration parameter p_1 recommendation.

closest k neighbors for the carrier, and then predicts the parameter value using the weighted average. k -nearest neighbors and simple cosine similarity based methods do not filter out the attributes that do not have a strong correlation with the configuration parameters. Thus, they suffer from inaccurate recommendations in cases where the carriers are highly similar but are labeled to be far away based on the distance influenced by the irrelevant attributes.

For enabling easy explanation of recommendations, we explore learners [20] such as decision tree and random forest. The decision tree learner starts from the carrier attribute observations, splits them on the feature that results in the largest information gain and thereby reducing the uncertainty in the final decision. This splitting process is repeated at each child node until the samples at each leaf node belong to the same class. The advantage of decision tree learner is simple explanation and understanding of the recommendations made for the configuration parameters. We show an example decision tree in Fig. 8. One can easily traverse down the tree following the attribute values and identify the parameter value. Decision tree provides explanations that our engineers found quite intuitive and easy to verify. The random forest learner uses ensemble techniques across multiple decisions trees and generates the average prediction of the individual trees. Ensemble techniques are known to control over-fitting and improve prediction accuracy.

Analogous to the motivation behind decision trees, we next explore collaborative filtering with chi-square tests of independence to build the dependency model and a simple voting approach across carriers to recommend the configuration parameter value that has highest support from other carriers. The chi-square test is a non-parametric test to identify the association between carrier attributes

		Configuration parameter			
		$p_1 = 20$	$p_1 = 40$	$p_1 = 100$	Total
Carrier attribute	Morphology = urban				$\sum_{b=1}^C (O_{1b})$
	Morphology = suburban				
	Morphology = rural				
	Total	$\sum_{a=1}^R (O_{a1})$			$\sum_{a=1}^R \sum_{b=1}^C (O_{ab})$

Figure 9: An example contingency table between attribute morphology and configuration p_1 .

and configuration parameters for existing carriers C_1, C_2, \dots, C_{N-1} . The test uses a contingency table to lay out the total counts for each pair of attribute value and configuration parameter value. Fig. 9 shows an example contingency table.

We compute the chi-square test statistic $\chi_{i,j}^2$ for each attribute X_j and each parameter $Y^{(i)}$ using

$$\chi_{i,j}^2 = \sum_{a=1}^R \sum_{b=1}^C \frac{(O_{ab} - E_{ab})^2}{E_{ab}} \quad (3)$$

where, O_{ab} is the observed count for the a^{th} row (attribute value) and b^{th} column (parameter value). E_{ab} is the expected cell count in the a^{th} row and b^{th} column of the table.

$$E_{ab} = \frac{\sum_{a=1}^R O_{ab} \cdot \sum_{b=1}^C O_{ab}}{\sum_{a=1}^R \sum_{b=1}^C O_{ab}} \quad (4)$$

R and C are the number of rows and columns. We then compare $\chi_{i,j}^2$ to the critical value from the chi-square distribution table with degrees of freedom $df = (R - 1)(C - 1)$ and selected confidence value. If $\chi_{i,j}^2$ is greater than the critical value, then we reject the null hypothesis that X_j and $Y^{(i)}$ are independent.

Thus, for each parameter $Y^{(i)}$, we identify the list of dependent attributes. For a new carrier C_N , we identify the similar carriers that have exact matching values on the dependent attributes. Amongst the similar carriers, we take a voting approach to identify the parameter value for each configuration parameter $Y_N^{(i)}$ that gets maximum support across the carriers. We use a threshold of 75% in our implementation and evaluation to determine the voting support.

Intuition behind why we believe collaborative filtering will better suit our problem: As described in Section 2.6, we observed a high variability or a large number of distinct values for the configuration parameters across different geographic locations, as well as high skewness in the distribution of the parameter values. This poses significant challenges for the classic classifiers such as decision tree, random forest, deep neural networks to accurately learn the dependency models. These classifiers typically aim to learn the models that best fit for the frequently occurring samples, and then treat the rare samples as outliers. It is equally important for us to accurately identify the parameter values that are configured for a small number of locations. Even though rare, these configuration parameter settings serve the purpose of providing enhanced service performance experience to the users. Thus, our goal is to accurately learn the dependency models for both the frequently as well as rarely occurring samples. Collaborative filtering and voting approach can handle this well (we will also demonstrate this using

empirical evaluation in Section 4). Collaborative filtering learns the dependency model for each parameter value specific to carriers and irrespective of whether the parameter value is rare or frequent across the network. It also eliminates the irrelevant attributes with respect to the parameter values. Models tuned to each carrier characteristic is thus expected to improve the prediction accuracy and recommend better configuration parameters for new carriers.

3.3 Geographical proximity

In Section 3.2, the dependency model in Auric uses all the existing carriers to learn the dependency model and then Auric recommends the configuration parameter value for the new carrier. We refer to these learners as *global learners* since they use the configurations across the whole network. In this section, we propose to adapt those learners to use the geographical proximity to restrict the carriers that can be used for learning and recommendation. When we apply geographical proximity, we refer to those learners as *local learners*. The intuition behind the local learners is that the configuration parameters on geographically nearby carriers are highly likely to have a strong dependency with the new carrier as opposed to far-away carriers. For example, a new carrier that is added in downtown Manhattan is likely to be similar in configuration to nearby carriers with similar attributes within downtown Manhattan as opposed to carriers far-away (say in Texas) even though they share similar attributes. The Texas configuration might be different because of different propagation patterns. Since the parameter tuning is highly local and the similarity diminishes as one goes farther away from the carriers, we expect the geographical proximity to improve the quality of recommendations.

In Auric, we use the X2 LTE neighbor relations to capture geographically nearby neighbors for the carriers. We thus employ collaborative filtering and voting approach with geographical proximity in Auric to recommend the configuration parameter values for the newly added carriers in the cellular network. We will show in our evaluation (Section 4) that geographical proximity plays a key role in configuration learning and the local learner outperforms the global learner in prediction accuracy. The local learners with decision tree, random forest, or deep neural network classifiers would still struggle to handle the high variability and skewness of the parameter values.

Our formulation and solution in Auric is general and can be easily extended to other networks and services. For example, when adding new node types such as routers, line cards, servers, 5G base stations, or even virtualized or containerized functions to the networks, one can leverage the data-driven approach from Auric to recommend configuration for the newly added nodes.

4 EVALUATION

In this section, we evaluate the accuracy of Auric in generating LTE network carrier configuration using data-driven learning and recommendation. Our evaluation is three fold: (i) identify the accuracy of our global learners, (ii) highlight the importance of geographical proximity, and (iii) validate our accuracy with the network engineers.

4.1 Data set

We use the configuration parameter data set collected from 400K+ carriers across 28 markets in a large LTE network as described in Section 2.6. A total of 65 configuration parameters with values consisting of a range form our predictee. In other words, these are candidate parameters which we would like to recommend the values for. 26 out of the 65 configuration parameters have to be set for pairs of carriers and their neighbors. These parameters are used to deal with user mobility and handovers across carriers. So, for each of the 26 pair-wise parameters, we represent the predictee variable using $Y_{j,k}^{(i)}$, which captures the i^{th} parameter for the current carrier j and its neighbor k . For the rest of the 39 singular parameters, we represent the predictee variable using $Y_j^{(i)}$ for each carrier j . Use Table 2 to review the notations.

We use the carrier attributes as our predictor variables (X) (refer to Table 1 for the list). For model learning and recommendation for singular parameters, we only use the attributes of the carrier, whereas for pair-wise parameters, we use both the attributes of the carriers and their corresponding neighbors. With 400K+ carriers, 26 pair-wise and 39 singular configuration parameters, we have a total of 15M+ configuration parameter values. This is a very large sample data set that we use for conducting our evaluation.

4.2 Methodology

Our evaluation approach treats each carrier like a new carrier of interest and uses the rest as the existing carriers for learning and recommendation. This gives us a large number of sample points (15M+) to generate the configuration, and then compare to the current values to identify if we recommended accurately or not. To begin with, we first evaluate the accuracy of global learners across only four markets with each one covering a different timezone. The intent is to identify if there is any global learner that significantly outperforms the others. Table 3 summarizes the timezone, number of carriers and eNodeBs, and configuration parameters for the four markets. The reason behind picking four out of 28 markets is to present an in-depth analysis for each market. For the 4 markets, we have a total of 4.5M configuration parameter values across 116K carriers. We use the standard machine learning cross-validation approach to compute the accuracy scores. We use scikit-learn for our global learners.

Table 3: Data set to compare global learners across four markets with each one covering a different timezone.

	Timezone	Carriers	eNodeBs	Parameters
Market 1	Mountain	24,271	1791	930,481
Market 2	Central	22,809	1521	676,627
Market 3	Eastern	45,127	2643	2,012,021
Market 4	Pacific	23,805	1679	909,010
All four		116,012	7634	4,528,139

1. **Decision tree learner.** We use Gini score to determine how to split and the tree is expanded until all leaves are pure (*i.e.*, all data points contain the same label).
2. **Random forest learner.** We use 100 trees in the forest, and Gini score for decision to split. Tree is expanded until all leaves are pure.
3. **k -Nearest neighbor learner.** We use $k = 5$, equal weighting across neighbors and distance metric of Euclidean.

4. **Deep neural network learner.** We use 7 hidden layers with sizes 100, 100, 100, 50, 50, 50, 10 tailored towards our input and output layer sizes. We use the adam solver which is a stochastic gradient-based optimizer, activation function of relu (rectified linear unit function - $f(x) = \max(0, x)$) for the hidden layers, the regularization L2 penalty of 10^{-5} , random state of 1, and maximum iteration of 10000.
5. **Collaborative filtering with chi-square test of independence.** We use a p-value of 0.01 which is the significance level to identify if the observed distribution matches the expected distribution. The recommendation is the parameter value that has support from at least 75% of the other carriers.

For each predictee vector $Y^{(i)}$ and predictor matrix X , we first perform one-hot encoding before we pass the data to the learner for learning the model. This is because a lot of our attributes and even configuration parameters take categorical values. So, if a vector x across all carriers takes value a , b , and c , then one-hot encoding will create three vectors $x = a$, $x = b$, and $x = c$, and the carrier with value b , will have three values as 0, 1, 0. The sum of the one-hot numeric array for a particular carrier should be equal to 1. One-hot encoding increases the number of variables in our classification problem.

In our global learner, we learn the model once and determine the prediction accuracy for the parameter of interest using all the remaining carriers. Accuracy can be intuitively viewed as the number of predictee configuration parameters whose recommendation, using our learner, matches the current configuration parameter value of the targeted new carrier of interest, divided by the total number of configuration parameters. We determine accuracy independently for each predictee variable Y^i across all carriers and their neighbors (for pair-wise type of parameters).

For our local learner, we geographically scope the remaining carriers to be within the 1-hop neighborhood based on X2 neighbor relations. In our second part of evaluation, we compare the global learner accuracy with that of the local learner for each of the four markets. We then expand our data set to the entire 400K+ carriers, 28 markets and 15M+ configuration parameter values and generate the configuration recommendations and quantify the accuracy for both global and local learners.

In our last part of evaluation through validation with engineers, our intent is to understand how does Auric perform across a very large number of carriers and their parameters. We take a few sample output and share those with the engineering teams across multiple markets to evaluate our recommendations that do not match the current configuration parameter values (we call these as *mis-matches*). Our goal is to discover if there are opportunities in improving our algorithms, carrier attributes, or network configurations.

4.3 Results

Our key takeaways from evaluation are: (a) collaborative filtering with chi-square test of independence and voting outperforms other global learners, (b) local learner with geographical proximity outperforms the global learner using collaborative filtering - this is because of how carrier configuration tuning is done by the engineers that have more local dependency with geographically nearby neighbors as opposed to far-away carriers, (c) a significant fraction

of the mis-matches (28%) result in improving the network configuration, thereby increasing our confidence in Auric's local learner, and (d) a small fraction of mis-matches indicate that we have to incorporate more carrier attributes that are representative of terrain type (e.g., facing mountain or tall buildings), signal propagations, or even recent configuration changes certified for a network-wide roll-out. This brings in new opportunities for potential future research.

4.3.1 Accuracy of global learners. Figs. 10a-10d show the configuration parameter prediction accuracy across five global learners for the four markets as described in Table 3. The X-axis show the 65 configuration parameters, Y-axis on the left shows the prediction or recommendation accuracy for five global learners and the secondary Y-axis shows the number of distinct values (or, variability) for each configuration parameter. We reverse sorted the list based on the distinct values. For all five global learners, the accuracy goes down when the variability (or the number of distinct values) goes up. This is shown towards the left half of the plots in Figs. 10a-10d. This indicates that configuration parameters with high variability become harder to recommend or predict. Configuration parameters with very low variability have similar accuracy for all global learners. Finally, we observe a correlation between learners across different configuration parameters. This implies that if prediction is hard for one, it is no different for other global learners.

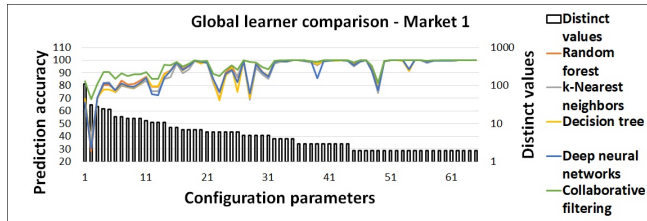
Table 4 summarizes the average accuracy across all configuration parameters for each global learner and across all four markets. Collaborative filtering with voting has a prediction accuracy of 95.48% and outperforms the other four learners because it can effectively handle the high variability and skewness of the configuration parameter values. Random forest learner performs slightly better than the rest of the three learners. Overall, a 90%+ accuracy is very encouraging for recommendation from the configuration perspective across all five learners.

4.3.2 Importance of geographical proximity. We now select collaborative filtering as our base global learner and compare it to its corresponding local learner with proximity defined using 1-hop X2 LTE neighbor relations. Collaborative filtering with local voting achieves a better prediction accuracy of 96.14% compared to 95.48% using collaborative filtering with global voting. We also generate prediction results by repeating the same process like above but now expanding from 4 markets to 28 markets with a total of 400K+ carriers and 15M+ configuration parameters. Collaborative filtering with local voting achieves a better prediction accuracy of 96.9% averaged across 28 markets as compared to 96.5% using collaborative filtering with global voting. A 0.4% improvement across 15M+ parameters accounts for 60K parameters that improve in prediction because of geographical proximity. Thus, this highlights its importance in the generation of carrier configuration.

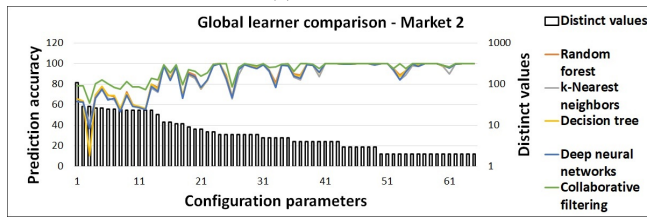
Next, we present results for the geographical proximity based recommendation for 4 out of the 65 configuration parameters that have the highest variability in Figs. 11a- 11d. The X-axis represents the 28 markets. The Y-axis on the left shows the prediction accuracy and the secondary Y-axis on the right captures the distinct value for configuration parameters in each market. One finding across the four charts is that different markets have different levels of variability in the parameters and thus the accuracy vary accordingly. Some markets (e.g., markets 6 and 7 in Fig.11a) have a lower prediction

Table 4: Average accuracy for five global learners across four markets and across all configuration parameters.

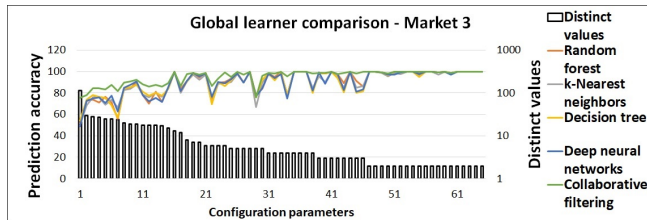
	Random forest	k-Nearest neighbors	Decision tree	Deep neural network	Collaborative filtering
Market 1	92.58	91.58	91.93	91.94	95.94
Market 2	89.27	88.08	88.73	88.39	93.75
Market 3	91.43	90.71	91.14	90.98	95.58
Market 4	95.15	94.34	94.79	94.57	96.63
All four	92.11	91.18	91.68	91.7	95.48



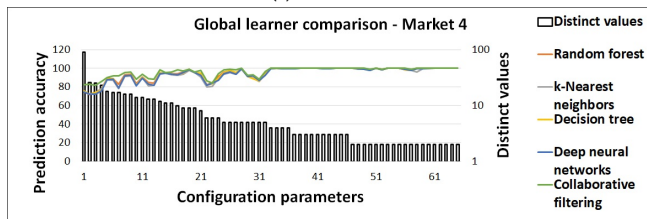
(a) Market 1.



(b) Market 2.



(c) Market 3.

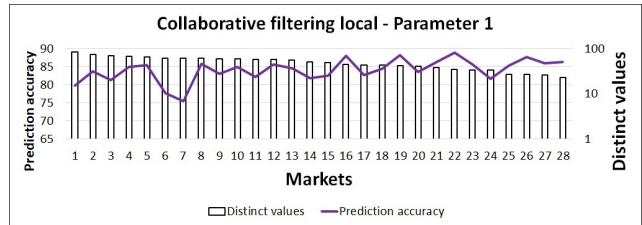


(d) Market 4.

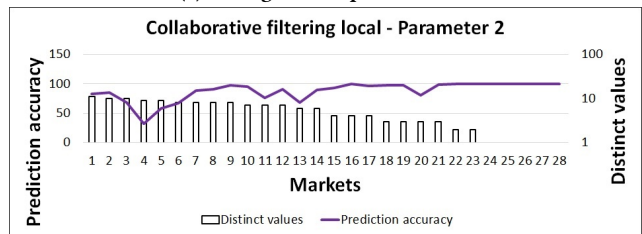
Figure 10: Prediction accuracy of five global learners for four markets.

accuracy even though their variability is similar to others. This implies that those markets could have some other attributes that we might be missing in our learners. This shows the challenge of tuning carrier configuration across different markets.

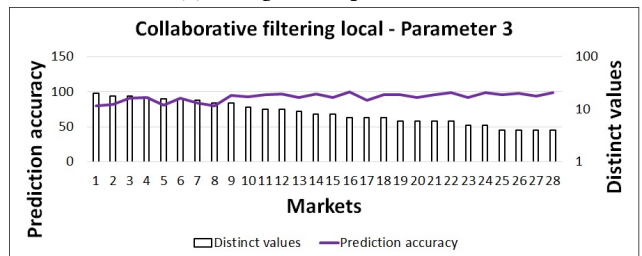
4.3.3 Accuracy validation with engineers. Approximately 96% is a good accuracy demonstration for geographical proximity based recommendation of configuration across 15M+ parameters. However, we were curious to understand why we got the recommendation not to match with the current configuration for the rest 4%. We shared these results with a few engineers across some of the markets and



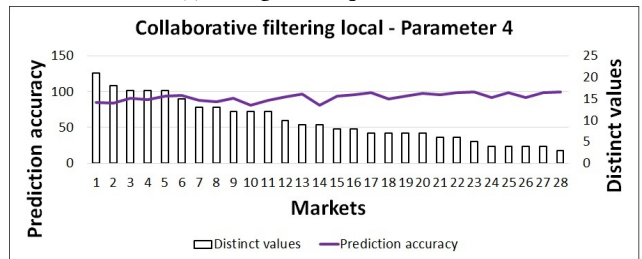
(a) Configuration parameter 1.



(b) Configuration parameter 2.



(c) Configuration parameter 3.



(d) Configuration parameter 4.

Figure 11: Prediction accuracy of local learner for four configuration parameters across all markets.

they were able to provide a labeling of the mismatch output. This analysis and labeling was conducted over multiple days to derive conclusions. We put the labels into three categories: (a) mismatches require update to our learner and/or our list of carrier attributes, (b) mismatches were indeed good recommendations that led to configuration changes to the network, and (c) need more time for investigation and thus currently labeled as inconclusive.

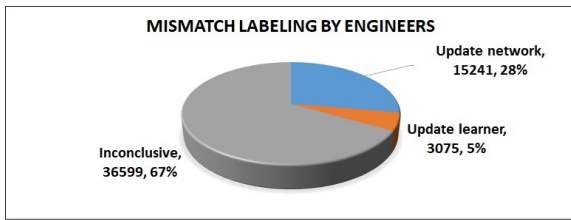


Figure 12: Labeling by the market engineers for the mismatches between the configuration parameter recommendation and the current values in the network.

Fig. 12 shows a pie-chart of the different labels. Out of a total of sampled 54,915 mis-matches, engineers labeled 3075 (5%) as update learner because they were not willing to change the current configuration parameter value. There were two reasons: (i) we were missing carrier attributes that were important for recommending the configuration parameter - e.g., terrain type and signal propagation; (ii) recent configuration changes on those carriers were part of an ongoing trial and certification of the network-wide roll-out and thus, they were not in the majority based on our voting approach. Both these learnings open up new opportunities to enhance our learner to bring in the temporal aspect of the configuration parameter changes, and derive carrier attributes from other measurements. As an example, by looking at user traffic and handover pattern, one could potentially derive that the carrier is facing a freeway, or time-of-day traffic behavior is indicative of a business or a residential location. Engineers labeled 15,241 (28%) as good recommendations and asked us to push those as configuration changes into the network. The reason for this mismatch was because the network had undergone some trials in the past and were left in a sub-optimal configuration.

The remaining 67% are under investigation at the time of writing this paper. After interactions with the network engineers, we learned that based on their experience and domain knowledge, they were not sure if the parameter change recommendations would result in any service performance improvements. Thus, they would first need to be trialed in some parts of the networks to observe their impacts, and then decisions can be made to roll it out across the whole network. We leave this as part of our ongoing work and plan to incorporate performance metrics as feedback to improve the learning and recommendation process. Hence, we currently labeled the 67% as inconclusive that warrant further experimentation and trial in the network.

Implications of inaccurate recommendations. After new carriers are added in cellular networks, the engineers carefully monitor the traffic distribution on the newly added carrier compared to neighboring existing carriers, and the service performance impact of the change (e.g., data throughput, voice call admissions). If they observe any unexpected performance impacts, they would immediately roll-back the configuration of the new carrier to its original default. Any inaccurate recommendation from Auric is treated similarly. In the future, we plan to automate the performance impact monitoring of recommended changes and any subsequent roll-backs.

5 OPERATIONAL EXPERIENCES

Based on our evaluation on a very large number of carriers and growing confidence with engineers, we deployed Auric in production environments to automatically generate configuration for newly added carriers. Auric is part of an end-to-end artificial intelligence based automation solution called SmartLaunch within the service provider to automatically launch new carriers. External vendors are contracted to do the physical work of integrating the new carrier into the network and its initial software configuration to set the parameter values. Once the carrier is ready for launch, the next step is to perform necessary pre-checks and then unlock the carrier, followed by monitoring alarms and key performance indicators as part of post-checks. These steps are automated within SmartLaunch and tremendously help the engineers to quickly launch the new carrier and have it carry user traffic.

We integrated Auric based auto generation and implementation of cellular configuration **before unlocking new carriers**. The change implementation before unlocking the carrier assists in cases where changing a parameter value requires the carrier to be locked (i.e., taken off-air). Locking a carrier (which is equivalent to a re-boot operation) after it is launched for implementing configuration changes has the risk of a minor disruption to traffic and service performance. Thus, in our current implementation, we conservatively avoid configuring such carriers to prevent any potential service disruption, and prefer pushing the configuration changes before unlocking the new carriers. Implementing the configurations recommended by Auric in production networks involves challenges akin to router configuration. Analogous to vendor-specific router configuration languages, cellular equipment vendors provide a configuration schema where the configuration parameters are organized in the form of a hierarchical structure called managed objects (similar to interfaces in routers). Further, equipment vendors provide an element management system (EMS) with interfaces at varying levels of sophistication (e.g. CLIs, APIs) for users to interact with the hardware. We implemented a controller that pushes the configuration recommendation from Auric into the network. Our controller compares the recommendations from Auric to the current configuration generated by the vendors and pushes only the mismatches. Mismatches can occur because of mistakes by vendors, out-of-date rulebooks, or pending tuning to be conducted to improve service performance. We allow an engineer to validate the mismatches before they are pushed into the new carriers. As the engineers gain confidence in Auric’s recommendations, the manual validation of mismatches becomes optional as part of the SmartLaunch automated solution. The controller maintains a vendor-specific template and automates the task of generating the configuration file by filling in the instance IDs from a database. The final configuration file is pushed through the EMS into the base station hardware.

Table 5: Auric operational experiences with new carrier launches. We intentionally show the results in part of the network for proprietary reasons.

New carriers launched	1251
Changes recommended by Auric	143 (11.4%)
Changes implemented successfully	114 (9%)

We report on two months of experience of running Auric for very large operational LTE networks. Table 5 shows the number

of carriers launched over the two month intervals. These results are only for the part of the network where engineers have adopted Auric. As you can see, out of 1251 new carrier launches, Auric recommends changes on around 143 carriers which is around 11.4%. Changes were successfully pushed on 114 out of 143 approved carriers. A total of 1102 configuration parameters were changed on those 114 carriers. We observed a small number of fall-outs (29) and the reasons were two fold: (a) Some engineers were prematurely unlocking the carriers through off-band interfaces and hence our current implementation would avoid making configuration changes for these carriers. This was intentional because we wanted to avoid any service disruption once a carrier is unlocked (*i.e.*, live and carrying traffic). In the future, we plan to address this by carefully pushing such configuration changes to a non-busy maintenance time window. (b) The configuration change implementation for some of the carriers resulted in timeouts because of the very large number of parameters and our setup based on EMS restrictions limited us in how many concurrent executions of parameters were supported. At the time of writing this paper, we are working with our internal teams to enhance our controller software to speed up execution for a large number of parameter changes.

Improvements to operational efficiency. Auric significantly reduced the time and effort to tune the configuration parameters for the carriers newly added to the cellular networks. Before Auric, the engineers would manually determine the parameter values based on experience, or tune them iteratively using trial-and-error approach to observe service performance improvements. The manual approach would take them several hours or days to converge to the near optimal configuration. Because Auric automatically learns and recommends configuration parameter values for the new carriers using dependency models derived from similar carriers in geographical proximity with matching attribute, we can converge to the near optimal parameter values faster. This resulted in significant improvements in operational efficiency.

Lessons learned. Interpretation of the configuration recommendation results and their simple explanation for engineers is extremely important for adoption and eventually aim for complete automation. Trust and interpretability are major challenges in adoption. Initially, the engineers were carefully validating the recommendations from Auric, and the performance impacts of the configuration changes. Through positive experience and increased confidence over time, Auric got integrated into an automated launch process when adding new carriers.

6 LIMITATIONS AND NEW OPPORTUNITIES

In this section, we discuss limitations of Auric and highlight new research opportunities for configuration management in networks.

Performance feedback for recommended configuration. Currently, Auric does not model the dynamic aspects of configuration, and performance feedback. This is because, for a new carrier which is in locked state and not yet carrying traffic, we could only leverage the static attributes of the carrier for recommending the configuration parameters. However, once it is unlocked and carrying live traffic, we could observe the traffic patterns on the new carrier and its neighbors, congestion scenarios, handovers due to user mobility, radio channel interference, and service performance impacts,

and then determine if we need to further tune the configuration parameters. It would be a very interesting research opportunity to explore the dependency model between configuration parameter values and their impacts on service performance and then use the learnings to enhance the configuration recommendations for newly added carriers as well as existing neighboring carriers. As an example, for the similar carriers with matching attributes and different distribution of parameter values, we can provide higher weights (in our voting approach) to configuration changes that have improved service performance in the past. The performance impacts for historical configuration changes can also potentially help us to address the inconclusive mismatches (67%) and guide the engineers in improving the configuration tuning process. Static attributes fit nicely into two-dimensional modeling as incorporated in Auric. With both static and dynamic attributes, the problem becomes multi-dimensional with configuration parameters and service performance modeled as a time-series. Sophisticated learning algorithms such as convolutional neural networks or recurrent neural networks could potentially help in improvising configuration recommendations.

Bootstrapping configuration for the unobserved. Since Auric uses a classification approach for data-driven configuration recommendation, it can only recommend parameter values associated with carrier attributes that have been historically observed in operational network data. Thus, if carriers are added with new attribute values (*e.g.*, new carrier frequency, or hardware version), then Auric cannot make recommendations because it would not be able to identify similar carriers with matching attribute values. In such scenarios, we currently stick with the default configuration settings which may not be optimal. The engineers then take a manual approach for tuning the configuration parameter values on these carriers. In the future, one could explore automated reinforcement learning based approaches to systematically trial new parameter values with performance feedback on the carriers.

7 CONCLUSIONS

Configuration management is a very important task for cellular network operations. In this paper, we proposed a new data-driven recommendation approach Auric to automatically generate configuration parameters for carriers added in the cellular networks. Auric uses geographical proximity and collaborative filtering to learn the dependency model between configuration and carrier attributes. Evaluation of Auric using real-world LTE network data shows a significantly high accuracy (96%). After verification of our results by the network engineers, we implemented changes that improved network configuration. Our deployment of Auric in production setup over two months and successful use by the engineering teams to update configuration for newly added carriers demonstrates its ability to effectively generate network configurations.

Acknowledgement

We thank our shepherd Kun Tan, ACM SIGCOMM anonymous reviewers, Jennifer Yates, Xuan Liu, and Chris Hristov for their insightful feedback on our paper. We appreciate the collaboration and continuous support from the Network Engineering and Operations teams in the application of Auric in production setup.

REFERENCES

- [1] 2015. 3GPP LTE TS 32.450. Telecommunication management; Key Performance Indicators (KPI) for Evolved Universal Terrestrial Radio Access Network (E-UTRAN): Definitions.
- [2] 2015. 3GPP LTE TS 32.500. Telecommunication management; Self-Organizing Networks (SON); Concepts and requirements.
- [3] 2015. 3GPP LTE TS 36.300. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description.
- [4] 2021. Firsnet. Retrieved January 26, 2021 from <https://www.firsn.net.com/>
- [5] 2021. Narrowband Internet of Things (NB-IoT). Retrieved January 26, 2021 from https://en.wikipedia.org/wiki/Narrowband_IoT
- [6] 2021. Symmetry, Skewness and Kurtosis. Retrieved June 14, 2021 from <https://www.real-statistics.com/descriptive-statistics/symmetry-skewness-kurtosis/>
- [7] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. AED: Incrementally Synthesizing Policy-Compliant and Manageable Configurations. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. Association for Computing Machinery, New York, NY, USA, 482–495. <https://doi.org/10.1145/3386367.3431304>
- [8] Deepak Agarwal and Bee-Chung Chen. 2009. Regression-Based Latent Factor Models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Paris, France) (KDD '09)*. Association for Computing Machinery, New York, NY, USA, 19–28. <https://doi.org/10.1145/1557019.1557029>
- [9] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (Los Angeles, CA, USA) (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 155–168. <https://doi.org/10.1145/3098822.3098834>
- [10] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2016. Don't Mind the Gap: Bridging Network-Wide Objectives and Device-Level Configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference (Florianopolis, Brazil) (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 328–341. <https://doi.org/10.1145/2934872.2934909>
- [11] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2017. Network Configuration Synthesis with Abstract Topologies. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (Barcelona, Spain) (PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 437–451. <https://doi.org/10.1145/3062341.3062367>
- [12] Rüdiger Birkner, Dana Drachler-Cohen, Laurent Vanbever, and Martin Vechev. 2020. Config2Spec: Mining Network Specifications from Network Configurations. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA.
- [13] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2018. Netcomplete: Practical Network-Wide Configuration Synthesis with Autocompletion. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (Renton, WA, USA) (NSDI'18)*. USENIX Association, USA, 579–594.
- [14] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (Savannah, GA, USA) (OSDI'16)*. USENIX Association, USA.
- [15] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A General Approach to Network Configuration Analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (Oakland, CA) (NSDI'15)*. USENIX Association, USA, 469–483.
- [16] Aaron Gember-Jacobson, Aditya Akella, Ratul Mahajan, and Hongqiang Harry Liu. 2017. Automatically Repairing Network Control Planes Using an Abstract Representation. In *Proceedings of the 26th Symposium on Operating Systems Principles (Shanghai, China) (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 359–373. <https://doi.org/10.1145/3132747.3132753>
- [17] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the 2016 ACM SIGCOMM Conference (Florianopolis, Brazil) (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 300–313. <https://doi.org/10.1145/2934872.2934876>
- [18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.
- [19] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference (Florianopolis, Brazil) (SIGCOMM '16)*. ACM, New York, NY, USA, 58–72. <https://doi.org/10.1145/2934872.2934891>
- [20] T. Hastie, R. Tibshirani, and J.H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer. <https://books.google.com/books?id=eBSgoAEACAAJ>
- [21] Yigong Hu, Gongqi Huang, and Peng Huang. 2020. Automated Reasoning and Detection of Specious Configuration in Large Systems with Symbolic Execution. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 719–734. <https://www.usenix.org/conference/osdi20/presentation/hu>
- [22] Siva Kesava Reddy Kakarla, Ryan Beckett, Behnaz Arzani, Todd Millstein, and George Varghese. 2020. GRoot: Proactive Verification of DNS Configurations. In *SIGCOMM 2020*. <https://www.microsoft.com/en-us/research/publication/groot-proactive-verification-of-dns-configurations/> Best Paper Award.
- [23] Siva Kesava Reddy Kakarla, Alan Tang, Ryan Beckett, Karthick Jayaraman, Todd Millstein, Yuval Tamir, and George Varghese. 2020. Finding Network Misconfigurations by Automatic Template Inference. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA. <https://www.usenix.org/conference/nsdi20/presentation/kakarla>
- [24] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (San Jose, CA) (NSDI'12)*. USENIX Association, USA.
- [25] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. 2014. Facing the cold start problem in recommender systems. *Expert Systems with Applications* 41, 4, Part 2 (2014), 2065 – 2073. <https://doi.org/10.1016/j.eswa.2013.09.005>
- [26] Hongqiang Harry Liu, Xin Wu, Wei Zhou, Weiguo Chen, Tao Wang, Hui Xu, Lei Zhou, Qing Ma, and Ming Zhang. 2018. Automatic Life Cycle Management of Network Configurations. In *Proceedings of the Afternoon Workshop on Self-Driving Networks (Budapest, Hungary) (SelfDN 2018)*. Association for Computing Machinery, New York, NY, USA, 29–35. <https://doi.org/10.1145/3229584.3229585>
- [27] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. 2011. Rapid detection of maintenance induced changes in service performance. In *ACM CoNEXT*.

- [28] Ajay Mahimkar, Zihui Ge, Jennifer Yates, Chris Hristov, Vincent Corrado, Shane Smith, Jing Xu, and Mark Stockert. 2013. Robust Assessment of Changes in Cellular Networks. In *ACM CoNEXT*.
- [29] Ajay Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. 2010. Detecting the Performance Impact of Upgrades in Large Operational Networks. In *ACM SIGCOMM*.
- [30] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. 2011. Debugging the Data Plane with Anteater. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 290–301.
- [31] Santhosh Prabhu, Kuan-Yen Chou, Ali Kheradmand, P. Brighten Godfrey, and Matthew Caesar. 2020. Plankton: Scalable network configuration verification through model checking. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA.
- [32] Chaithan Prakash, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, Puneet Sharma, and Ying Zhang. 2015. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (London, United Kingdom) (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 29–42. <https://doi.org/10.1145/2785956.2787506>
- [33] Juan Ramiro and Khalid Hamied. 2011. Self-Organizing Networks (SON): Self-Planning, Self-Optimization and Self-Healing for GSM, UMTS and LTE. (2011).
- [34] Al Mamunur Rashid, George Karypis, and John Riedl. 2008. Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach. *SIGKDD Explor. Newsl.* 10, 2, 90–100. <https://doi.org/10.1145/1540276.1540302>
- [35] Shambwaditya Saha, Santhosh Prabhu, and P. Madhusudan. 2015. NetGen: Synthesizing Data-Plane Configurations for Network Policies. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (Santa Clara, California) (SOSR '15)*. Association for Computing Machinery, New York, NY, USA, Article 17, 6 pages. <https://doi.org/10.1145/2774993.2775006>
- [36] Jesus Bobadilla Sancho, Fernando Ortega Requena, Antonio Hernando Esteban, and Jesús Bernal Bermúdez. 2012. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems* 26 (February 2012), 225–238. <http://oa.upm.es/15302/>
- [37] Samuel Steffen, Timon Gehr, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2020. Probabilistic Verification of Network Configurations. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (Virtual Event, USA) (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 750–764. <https://doi.org/10.1145/3387514.3405900>
- [38] Xudong Sun, Runxiang Cheng, Jianyan Chen, Elaine Ang, Owolabi Legunsen, and Tianyin Xu. 2020. Testing Configuration Changes in Context to Prevent Production Failures. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 735–751. <https://www.usenix.org/conference/osdi20/presentation/sun>
- [39] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky H.Y. Wong, and Hongyi Zeng. 2016. Robotron: Top-down Network Management at Facebook Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (Florianopolis, Brazil) (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 426–439. <https://doi.org/10.1145/2934872.2934874>
- [40] Chunqiang Tang, Thawan Kooburat, Pradeep Venkatachalam, Akshay Chander, Zhe Wen, Aravind Narayanan, Patrick Dowell, and Robert Karl. 2015. Holistic Configuration Management at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles (Monterey, California) (SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 328–343. <https://doi.org/10.1145/2815400.2815401>
- [41] Bingchuan Tian, Xinyi Zhang, Ennan Zhai, Hongqiang Harry Liu, Qiaobo Ye, Chunsheng Wang, Xin Wu, Zhiming Ji, Yihong Sang, Ming Zhang, Da Yu, Chen Tian, Haitao Zheng, and Ben Y. Zhao. 2019. Safely and Automatically Updating In-Network ACL Configurations with Intent Language. In *Proceedings of the ACM Special Interest Group on Data Communication (Beijing, China) (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 214–226. <https://doi.org/10.1145/3341302.3342088>