Summarizing Network Configuration Patterns

Jyotirmay Chauhan*
Colgate University

Aaron Gember-Jacobson
Colgate University

1 Introduction

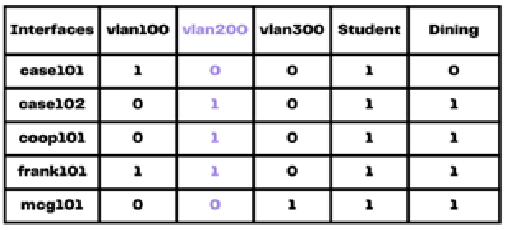
Configurations are a crucial but error-prone component of networks. While significant progress has been made in efficiently verifying network configurations using various forms of model checking, correctly modeling configuration semantics remains a complex and error-prone task [3, 8]. Enumerating policies to check is also hard, and using policy-mining tools [4] leads to a conundrum where configurations are verified based on polices inferred from those configurations.

An alternative verification strategy is to infer patterns from raw configuration syntax [6, 7], thereby avoiding the difficult tasks of modeling configuration semantics and enumerating policies. Identifying patterns within and across configuration elements allows network engineers to find pattern violations and flag them for further review. Minerals [7] and SelfStarter [6] follow this philosophy and identify similarities (and discrepancies) in configurations. However, these tools focus their attention on identifying relationships between a few select elements—e.g., interfaces [7] or access control lists (ACLs) [6]—while ignoring other important elements such as layer-2 components, syntactic sugar, and comments.

In prior work [5], we proposed using Contrast Set Learning (CSL)—a form of association rule mining which attempts to identify meaningful differences between separate groups—to identify subtle but important patterns in network configurations. For example, the presence/absence of the keyword "student" in a interface's description is a meaningful difference when interfaces with this keyword participate in a certain VLAN, whereas interfaces without this keyword do not participate in the VLAN. Deviations from this pattern are indicative of potential security or reachability problems.

This approach also allows for transparency in the decision-making process, which, given the vital nature of configurations, is a significant advantage over black-box machine learning models. However, since CSL's output scales exponentially with the size and complexity of the configurations, discerning meaningful relationships remains challenging. This poster introduces a rule filtering, selecting, and condensing process to address this challenge.

*Undergraduate student author



Interfaces	vlan100	vlan200	vlan300	Student	Dining
case101	1	0	0	1	0
case102	0	1	0	1	1
coop101	0	1	0	1	1
frank101	1	1	0	1	1
mcg101	0	0	1	1	1

Figure 1: A sample CSL input for Small university network. Interfaces is the primary key and vlan200 is the group feature

2 Our Approach

CSL Input. CSL takes as input a dataset, a group feature, and a rule length. The dataset is a tabular representation of the networks' configurations which encapsulates the relationships between different elements. We leverage existing approaches [1] to identify configuration elements and their relationships with other elements. The group feature is the target of our rule-generation—CSL attempts to identify meaningful differences (using other network attributes) between different values of the group feature. Rule length is the number of network attributes involved in predicting a particular group feature value.

Rule Generation. CSL ordinarily outputs every possible rule (of stated length) for each unique group-feature value satisfying a minimum threshold of support. We modified an open source implementation of the CSL algorithm STUCCO [2] to eliminate this threshold requirement and obtain all possible rules to widen our pools for meaningful rule search. Each rule is an IF-THEN statement about the group-feature value—IF part containing a combination of network attribute-value pairs(limited by rule length) and the THEN part pointing to the group-feature value it predicts. Since, STUCCO only produces rule for a particular group-feature, we repeat this process for all possible group-features. This process scales exponentially with more group-features indicating a need for faster rule generation. One of our contributions is parallelizing several segments of STUCCO to reduce the computation by several orders of magnitude.

Rule Filtering. A good rule set should contain rules covering all possible instances of a chosen group-feature value while being limited enough for proper verification. Simple filtering mechanism based on precision, recall, and f1 score were unable to isolate useful rules. We reattained precision as a metric because we want our our rules to be accurately capture underlying patterns. Recall became unimportant because we could use multiple rules for predicting different instance of same group-feature value —this does away with the necessity of broadly applicable rules. This led us to develop a new metric for rules called Rule coverage, which identifies all the instances where both the IF and THEN part of a particular rule are satisfied. This metric proved to be critical in building comprehensive rule-set while controlling for overlap. Overlap, for an instance of group-feature value, is defined as the number of rule-sets covering it. While some degree of overlap is preferable, a high degree of overlap in the rule-set suggest redundancy in the rule set. Thus precision in conjunction with Rule Coverage allows us to reduce the STUCCO output to small set of of useful rules using a greedy heuristic to select rule-sets while controlling the the overlap.

Next-Best Rule Set. Next best rule selection is a greedy heuristic which selects the rule with most coverage in each step. Since current coverage depends on the previous rules selected (and their coverage), a greedy approach allows us to maximize total coverage with the fewest possible rules.

It works by creating a two dimensional matrix for each group-feature. Each column is a rule produced by STUCCO while each row matches a corresponding instance of the group-feature value in the initial database. The matrix is populated using the rule coverage metric with 1 if there is rule covers the row otherwise 0 if there is none.

$$C_{\text{group}} = \begin{array}{cc} & \textit{Rules} \implies \\ \begin{array}{c} \textit{Rows} \\ \downarrow \end{array} & \begin{pmatrix} 1 & 0 & 1 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \end{pmatrix} \end{array}$$

We introduce the FOR_SUM vector $A \Longrightarrow [k_1, k_2, k_3 ... k_n]$, where k_i is the initial row weight (a hyperparameter) and n the length is the number of rows. Each element tracks the current coverage of the corresponding row in the original dataset. For simplicity, we assume $k_i = k \; \forall \; i$ such that $1 \leq i \leq n$, i.e. each row has the same weight at the beginning.

Finally, the FOR_REDUCTION $B \Longrightarrow 1/d$ is a hyperparameter for controlling rule overlap: a higher value of d corresponds to more overlap and vice versa.

To select the next rule, we multiply the FOR_SUM with the two dimensional matrix and select the rule with the highest value. We append this rule in to our final rule-set, and multiply the corresponding column vector by the FOR_reduction value, substituting each 0 with 1 and dot multiply it with FOR_SUM matrix. This process is repeated until the sum of all elements in FOR_SUM approaches 0.

Rule Set Condensation. For each group-feature, we identify the most impactful elements from the final rule set. Since high impact network attributes-value pairs tend to repeat themselves in multiple rules in the final rule-sets, we group the rules together according to their recurrence. The rules containing the most recurring element are grouped together under a umbrella group; we identify the most recurring element in the remaining rules and repeat the process. These process creates a hierarchy within the final rule set allowing for further insight in a rule's relative impact on the group-feature value.

3 Preliminary Results & Future work

We implemented our approach in python and evaluated it using configurations from small university campus. We limited our search to two length rules and produced condensed rule sets for the network-wide configurations. Our final rule-sets are significantly smaller than the STUCCO output rules and contain rules deemed useful by the network engineers. Our future work will be directed toward: i) more comprehensive testing in other enterprise networks; ii) Presenting violations in more user friendly interactive graph. iii) Comparing the efficacy of different length of rule sets.

References

[1] S. Alam, D. Lee, and A. Gember-Jacobson. Poster: Identifying syntactic motifs and errors in router configurations using graphs. In *International Conf. on Network Protocols (ICNP)*, 2022.

[2] S. D. Bay and M. J. Pazzani. Detecting change in categorical data: Mining contrast sets. In *KDD*, 1999.

[3] R. Birkner, T. Brodmann, P. Tsankov, L. Vanbever, and M. T. Vechev. Metha: Network verifiers need to be correct too! In *NSDI*, 2021.

[4] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. T. Vechev. Config2spec: Mining network specifications from network configurations. In *NSDI*, 2020.

[5] J. Chauhan, D. Lee, E. Yu, and A. Gember-Jacobson. Detecting configuration errors via pattern mining. In *Network Verification Workshop (NetVerify)*, 2021.

[6] S. K. R. K., A. Tang, R. Beckett, K. Jayaraman, T. D. Millstein, Y. Tamir, and G. Varghese. Finding network misconfigurations by automatic template inference. In *NSDI*, 2020.

[7] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb. Detecting network-wide and router-specific misconfigurations through data mining. *IEEE/ACM Trans. Netw.*, 17(1):66–79, 2009.

[8] F. Ye, D. Yu, E. Zhai, H. H. Liu, B. Tian, Q. Ye, C. Wang, X. Wu, T. Guo, C. Jin, et al. Accuracy, scalability, coverage: A practical configuration verifier on a global wan. In *SIGCOMM*, 2020.