WRS: Workflow Retrieval System for Cloud Automatic Remediation

Hongyi Huang
Tsinghua University
hhy.hongyi@outlook.com

Wenfei Wu
Peking University
wenfeiwu@pku.edu.cn

Shimin Tao
Huawei
taoshimin@huawei.com

***Abstract*—Remediation systems in modern information infrastructure gradually take over the increasing workload of unexpected events from operators. The core logic in such systems — remediation rule <symptom, workflow> — still needs operators to fill in manually, which is a tedious and error-prone process. We propose Workflow Retrieval System (a.k.a. WRS), which helps the operator to build remediation rules.**

WRS is a *recommendation system* that recommends structures from existing rules to operators when they are creating new ones. WRS formalizes the workflows in remediation rules as trees and extracts representative atomic structures from them. Then WRS organizes atomic structures in two ways to accelerate the later retrieval — a two-level hierarchy which helps searching similar structures, and a keyword indexing structure which helps searching by words. With the two retrieval structures, we build two applications: one is real-time workflow auto completion application which recommends remaining structures based on the existing partial structure, and another is workflow recommendation where WRS recommends atomic structures based on the description of the remediation symptoms.

We prototype WRS and evaluate it with legacy device vendors' operation manual. Our evaluation shows that WRS reasonably extracts and organizes atomic structures in remediation workflows, and the two applications atop it show fast execution time and high accuracy.

I. Introduction

Modern cloud management requires operators to stay on call for its service availability in 24/7. When unexpected events (e.g., outage, client tickets) happen, the operators would take actions to *remedy* the system — either recover the system to a healthy state or mitigate the unhealthy symptom to keep the service surviving.

Traditional infrastructure management usually trains operators with an operation manual, which consists of *remediation rules* such as "if a symptom X happens, take the action of A". With the size and the complexity of cloud infrastructure increasing, such manual operation is proposed to be overtaken by *automatic remediation systems*[32]. StackStorm[8], Rundeck[11] and FBAR[10] are examples of such systems, where remediation rules are stored in *remediation databases*.

However, the process of switching from manual operation to automatic remediation is progressing slowly. According to our survey (§II) with cloud operators, an important reason is that populating the remediation rule database is tedious and error-prone. Inexperienced operators could possibly build wrong remediation rules, causing disastrous results; [41] and even experienced operators need to spend huge effort to build rules for immense exceptional events (e.g., a switch can have more than 1000 error codes[9]).

We propose a Workflow Retrieval System named WRS to *accelerate the process of building remediation rules*. It is a recommendation system that targets the scenario where there exist a few remediation rules in the database, and the operator would like to add new rules to cover more events or devices.

WRS formalizes a remediation rule as a tuple <symptom, workflow, description>; a workflow is a tree describing the execution flow to recover the system or mitigate the failure. WRS refines the remediation database in three ways which improve the accuracy and efficiency in later recommendations. (1) WRS extracts atomic structures in workflows instead of storing whole workflows, which is more concise, accurate, and representative. (2) WRS builds a two-level hierarchical searching structure to organize all atomic structures, which can quickly locate a specific atomic structure during retrieval. (3) WRS builds a keyword indexing system to fetch semantic-related atomic structures, which accelerates the atomic structure lookup in later recommendations.

We overcome a challenge in WRS. In atomic structure extraction and the hierarchical retrieval structure construction, a measure to compute the distance of two atomic structures needs to be defined. WRS combines the semantic similarity in natural language processing (NLP) and the tree edit distance algorithm (TED) in graph theory to define the distance measure in WRS.

Atop WRS, we build two applications to validate its usefulness. First, when a workflow is partially constructed, the application can search the hierarchical retrieval system and return similar structures as the reference for the remaining part. Second, when a workflow's description is written, the application could search the keyword indexing system and recommend atomic structures to the operator.

We build a dataset using a network device vendor's manuals of five device models. We prototype WRS and measure its processing speed in retrieving atomic structures and its accuracy in two recommendation applications. The evaluation shows WRS can serve the infrastructure operators in user-acceptable real-time (less than 4 seconds) and high accuracy (95% at most). Our contributions in this paper are as follows.

Wenfei Wu is the corresponding author.

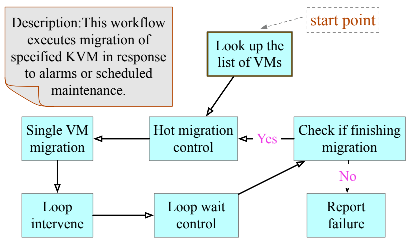

Description:This workflow executes migration of specified KVM in response to alarms or scheduled maintenance.
start point
Look up the list of VMs
Single VM migration
Hot migration control
Yes
Check if finishing migration
No
Loop intervene
Loop wait control
Report failure

Fig. 1: VM Migration Workflow in a Remediation System

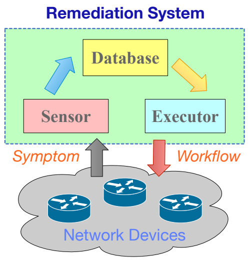


Fig. 2: Architecture of a Remediation System

- We formalize the basic structure of cloud remediation rules and propose WRS to extract and organize atomic structure in the workflow of remediation rules, which could improve the accuracy and speed in recommendation applications.
- We build two applications atop WRS, which could recommend operators new rules during the rule construction.
- We build a dataset based on device manuals to validate WRS. And we opensource WRS and our dataset for the community to use.

II. Background

A. Cloud Remediation

Remediation Examples. TABLE I shows a snippet of a network device vendor's operation manual. The operator may get an alarm (symptom) saying "there is bursty traffic in the device". There may be two root causes: the device itself has insufficient cache or the connected host devices input excessive traffic. The actual diagnostic steps for the operator would be as follows. First, check whether the BE queue drops packet — if yes, it is one of the two root causes, otherwise, it is none of them; Second, check if RTN uses the smallest cache — if yes, the root cause is insufficient cache, otherwise, it is excessive burst from hosts.

Fig.1 shows a workflow of "VM hot migration". This is a simplified automatic process implemented in StackStorm in the author(s)' organization. It takes a few sequential steps to migrate one VM, and then checks whether the migration finishes; if yes, it proceeds with the next one; otherwise, reports failure.

The third example is from the experience of a campus network operator. When the operator gets a report of "computers in office A cannot connect to the Internet", the operator would first ping an Internet website. If the ping gets through, the operator would further check security rules such as allow/deny list of TCP ports; otherwise, the operator would check the routing protocols and routing tables.

With the examples above, we summarize that a few characteristics of remediation.

- It is usually triggered by an event with a symptom (either fault alarm or maintenance).
- A remediation starts from a single root. The reason is that human operators must take a first-step action to start the diagnosis. [41]
- Each step of the workflow executes a command. It can be a command without output, and then the workflow proceeds to its successor step; it can also be a command with several possible results, used to choose the next step. In most remediation processes, the command would be supplemented with human readable description. [41]
- Most remediation flows have a tree structure — starting from the root, executing a command, choosing one of the branches (next step) according to the result of the current step, and proceeding until the problem is solved. If the workflow contains circles, we can easily break the circle at the edge from the child to an ancestor and get a tree.
- The remediation usually goes with a description in natural language explaining the symptom (e.g., error code) and possible causes[38]. For example, operation manuals and expert querying systems usually have such descriptions[9].

Remediation Rules. We define a remediation rule as a 3-tuple: <symptom, workflow, description>. The symptom is usually the reason that triggers a workflow to remedy the system, and the workflow is a tree-structure that describes the execution plan to automatically or manually solve/mitigate the problem. And the description is human-readable text that helps the operator to understand the root cause and the maintenance operations.

Remediation Systems. Cloud operators have been starting to apply automatic remediation systems in system failure recovery/mitigation. For example, StackStorm[8] and Rundeck[11] are two platforms that integrate heterogeneous infrastructure management systems; and FBAR[10] is a set of daemons "execute code automatically in response to detected software and hardware failures".

Fig. 2 shows the architecture of a remediation system, which consists of a sensor to get the symptom of infrastructure problems, a database engine to make decisions, and an executor to run a workflow to the infrastructure. The database contains remediation rules and chooses workflows according to the symptom.

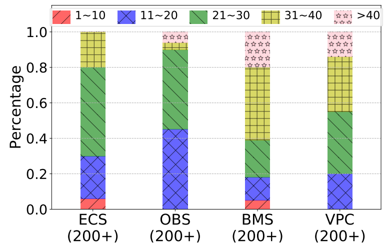
B. Problem in Transferring to Automatic Remediation

Difficulty in Deployment. While the remediation system/platform is ready, the switching from manual maintenance to automatic remediation is still progressing slowly [23]. We conducted a survey with 27 operators in a cloud provider and a university IT department, studying the complexity in building and maintaining the remediation systems. Part of the results are shown in Fig. 3.

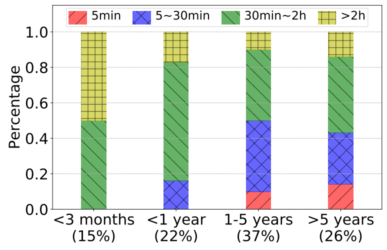
Fig. 3a shows the number and the size of workflows estimated by the respondent in a cloud. Among the four businesses we studied (Elastic Cloud Server, Object Storage, Bare Mental Server, Virtual Private Cluster), all of them have more than 200 workflows in daily operation, and the average size of their workflows ranges from 22 to 31. From Fig. 3b, in daily operation, the experience of operators determines the time they spent in fixing bugs in the workflows. For example, less experienced operators (<3 months) need 97 minutes to debug a workflow. Fig. 3c shows the issues that lead to fault rules in the rule creation. Most faulty rules are created due to the operators being unable to design a workflow correctly

TABLE I: Runbook Snippet from a Vendor's Device

BUSINESS	ALARM	ROOT CAUSE	DIAGNOSTIC PROCESS
PKT-Eline	Burst Traffic	Insufficient cache	1. check whether BE queue drops packets from performance statistics in ports; 2. check if RTN uses the board with smallest cache
		Excessive burst for host devices	1. check whether BE queue drops packets from performance statistics in ports; 2. check if RTN uses the board with largest cache



(a) The Number and the Size of Workflows that Operators Need to Orchestrate by Hand



(b) The Debugging Time with Regard to the Experience of Operators

Cause	%
problem not defined by workflow	14
implementing the workflow on the specific platform incorrectly	21
insufficient support for the workflow functionality on certain platforms	26
workflow is designed incorrectly	39

(c) The Proportion of Observed Rationales to Blame for the Bugs

Fig. 3: Results of the Survey with 27 Network Operators

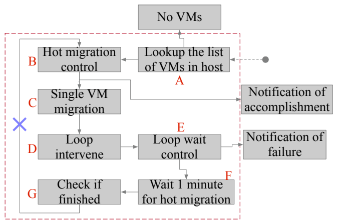


Fig. 4: Workflow: KVM Migration

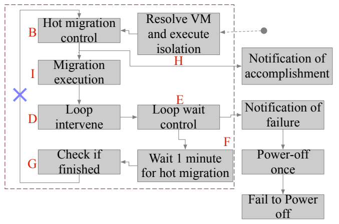


Fig. 5: Workflow: File System Migration

(39%), and others are implementation specific or platform constrained.

From the study, we find that the workflows have their own complexity (e.g., a typical switch can have 1000+ error codes), and inexperienced operators are more easily building faulty workflows. The heavy workload, the tedious rule creation process, and the possibly disastrous results together lead to the slow progress of rule breeding in remediation system.

Workflow Recommendation System. To accelerate the workflow construction, workflow recommendation systems are proposed, which aids the operator in incrementally enriching the remediation database. When an operator decides to build a remediation rule, the system would provide related workflows to the operator as a reference, and the operator can reuse existing workflows to build a new rule. And thus, the process of building the database can be accelerated.

For example, Fig. 4 and Fig. 5 show two workflows (i.e., KVM migration and file system automatic migration) in a production cloud. They both depict the core operations that conduct the migration of the objects. If the recommendation system can provide one rule as a reference to the operator to build another one, the new rule can be built more quickly and accurately with copy and paste.

Existing Practice (Expert System). A common practice for large infrastructure providers is to build an *expert system*, which stores existing remediation rules in raw format, and after an operator inquires about a symptom, the raw data is returned. An operator can ask the system "how to operate when symptom X happens?", and the system would use this query to search the database and find and return rules whose descriptions are semantically close to the query (e.g., keywords matching). And it's tedious for operators to look through unstructured raw data.

Moreover, expert systems still need improvement in terms of recommendation accuracy. Events and workflows are heterogeneous (e.g., KVM and FS migration) and different in granularity (e.g., disconnection to the Internet and to the campus gateway), and the workflow can hardly be exactly the same. Thus, providing the raw workflow might be inaccurate. It may miss critical actions (too small) or contain unnecessary actions (too large).

C. Goal and Intuition

Our goal in this paper is to build a system that can *recommend workflows* according to the operator's query, and it had better answer the query *accurately and quickly*. We observe that existing workflows usually have common structures that can be reused when building new ones, which we call *atomic structures*. If atomic structures can be extracted from raw workflows, they would be more representative to stand for a process (providing accuracy), and their small size can accelerate the search process (providing quick response). Looking into the examples in Fig. 4 and Fig. 5, we can find such atomic structures (illustrated in the dashed boxes), which are similar among workflows.

III. Design

A. Overview

We design a system named WRS atop the remediation database and used by the operator. WRS refines the raw remediation rules into intermediate formats and provides interfaces for operators to query. Fig. 6 shows the architecture of WRS. Like an expert system, WRS is a computer-aided tool for operators to query and get recommendations from the existing remediation database, and the operator makes the customization according to the recommendation to enrich the database.

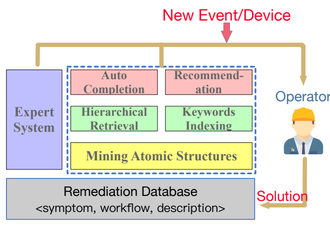


Fig. 6: Overview of WRS

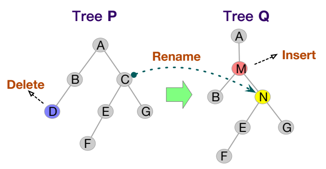


Fig. 7: Tree Edit Distance

WRS formalizes the data model of a workflow. And then, it extracts atomic structures among workflows, which are representative to denote a set of actions (§III-B). WRS organizes atomic structures in two ways, which accelerates the later retrieval process: the first one would classify "similar" atomic structures into groups and choose on a representative structure to stand for the group, which would accelerate the structure auto completion application (§III-C); the second one would index atomic structures by keywords in their description, which is used in workflow recommendation when an operator uses the symptom description to search the database (§III-C).

B. Mining Atomic Structures

We first present the data model of a workflow. To mine representative structures, we need a measure of the distance of structures; we choose tree edit distance (TED) and customize the measure according to the characteristics of workflow. Finally, we present the algorithm of atomic structure mining.

Data Model of Workflow. According to the observation in II, a workflow is represented in a tree structure; if not a tree (e.g., Fig. 4 and Fig. 5), the inner circle would be broken by concealing the reverse edge (an edge from a node far from the root to a node near the root, depicted as the cross in the figures) to be utilized later. Each node in the tree contains a command/action to execute (to a device) and a description to explain the purpose or the functionality of the command. Each node may have directed edges to its children — if the command of the node has no output (e.g., reboot), the node cloud have one edge point to its child (executing the next command); if the command of the node has an output (e.g., checking CPU), it may have one or multiple edges to its children, with each describing one possible result of the output and its corresponding next step (e.g., CPU high/medium/low).

Tree Edit Distance. To find common atomic structures, we need to define the *distance* between structures. A commonly used measure is the Tree Edit Distance (TED). We customize the TED algorithm for workflows.

In TED algorithms, the node on trees has a label (e.g., characters in Fig. 4 and Fig. 5)); the distance of two nodes is defined as 1 if their labels are different and otherwise 0. Fig. 7 shows an example to compute TED. To transform the tree P to the tree Q, three operations are necessary, deleting node D to tree P, inserting node M to tree Q, and renaming node N (on Q) to C. With the cost of each operation (deleting, inserting, and renaming) defined as one, the total cost of making P and Q equal is three, which is also the distance between them.

Algorithm 1 SS-TED, Cited from [34]

```
Result: semantic similarity of tree pair
1: $\delta(\phi, \phi) = 0$
2: $\delta(F, \phi) = \delta(F - v, \phi) + c_d(v)$
3: $\delta(\phi, G) = \delta(\phi, G - \omega) + c_i(\omega)$
4: if F is not a tree || G is not a tree then
5:     $\delta(F, G) = min(\delta(F-v, G)+c_d(v), \delta(F, G-w)+c_i(w), \delta(F_v, G_w)+ \delta(F - F_v, G - Gw))$
6: end if
7: if F is a tree && G is a tree then
8:     $\delta(F, G) = min(\delta(F - v, G) + c_d(v), \delta(F, G - w) + c_i(w), \delta(F - v, G - w) + c_r(v, w))$
9: end if
```

The basic TED algorithm is shown in Algorithm 1. The algorithm takes two forests (including trees) as input, recursively searches all possibilities to make the two inputs equal, and returns the minimum cost in the whole searching. If one input is empty, the cost is the size of another input. If one input is not a tree (a forest), the algorithm proceeds to search two cases (and return the minimum cost) — remove one node from one input (cost one) and search the remaining two forests or take two trees out from both inputs and search the two trees and the remaining two forests. If both inputs are trees, the algorithm proceeds to search two cases — remove one node from one tree (cost one) and rename both trees' root nodes (cost zero).

Customize the Distance Measure. The original TED algorithm cannot be directly applied for workflow comparison, and we make customization in two aspects. First, we break loops by removing the minimum number of edges: WRS applies width-first search to visit all edges and collect edges to form a spanning tree; if an edge would cause loops to the spanning tree, it is abandoned; otherwise, it is added to the spanning tree.

Second, the measure of the distance between two nodes in TED is either 0 or 1, indicating different labels or the same label, which does not apply to the nodes in workflows. In a workflow, a node represents a command to an entity, which is not binary 0 and 1; it is quite possible that two different entities can be applied with similar operations. For example, the "Single VM migration" in Fig. 4 and the "Migration execution" in Fig. 5 are two similar actions to VMs and File Systems except for the objects, but their text description is not completely the same. It is more reasonable to compare their "semantic similarity".

We transform workflow nodes into word embedding formats and use their distance in the word embedding space as the cost to change between each other. To achieve this, WRS first integrates with a word2vec[20] neural network model and feeds a node's description (or command) into the model. The model would output a sentence vector in the word embedding space. For two nodes, their similarity is the cosine value of their vectors, i.e., $(\vec{v_1} \cdot \vec{v_2})/(|\vec{v_1}| \cdot |\vec{v_2}|)$. And we define the distance as $1 - similarity$.

In the WRS customized TED algorithm, the cost of insertion and deletion is still one, but the cost of rename is the distance

$$cost(rename) = 1 - similarity.$$

The overall customized TED algorithm is similar to Algorithm 1 except c_r defined as the "cost" above. We name the customized algorithm as SS-TED.

Atomic Structure Mining Algorithm. WRS first enumerates all subtrees of all workflows, and then compares the pairwise distance between all pairs of subtrees. The pair whose distance with SS-TED is within a threshold K is regarded as a similar pair. All subtrees in similar pairs are recorded as *atomic structures*, because each of them is at least representative for a pair.

The distance threshold K needs to be tuned to a moderate value. A small value of K would make the mining algorithm return less number of atomic structures, but they are more alike; a large value of K is vice versa. We would tune the parameter in §V because both the number and the precision of mined atomic structures are fundamental to top-level applications.

We further apply two optimizations in the mining algorithm in WRS. First, it is possible that within a pair, one subtree is the subtree of the other. WRS applies a filter to check the relationship and keeps the larger subtree. Second, there are a few cases where a workflow's size is too large, causing a large number of subtrees, which would cause the pairwise comparison to be of low efficiency. Hence, we make two-round mining to the workflow set. In the first round, we only apply the mining algorithm to small workflows and get the representative atomic structure, and then in the second round, we remove the atomic structures from the large workflows (if they contain any) and decompose the remaining part to subtrees and apply the mining algorithm again to all subtrees in both rounds. We then merge atomic structures as the first step, if possible, to only keep the largest ones.

Example. In the example of Fig. 4 and Fig. 5, WRS would enumerate all subtrees in both workflows and return pairs whose similarity is larger than a threshold. WRS would return atomic structures such as E-F-G, B-C(I)-D in both workflows.

The dashed box is the largest subtree, and the distance of that in both workflows is computed as follows. Among all node pairs, A and C need to be renamed. The semantic similarity of A and H, C and I on both workflows are 0.52, 0.73 respectively, so the costs are 0.48, 0.27. Thus, the distance between the two structures in both workflows is 0.48+0.27=0.75.

Complexity. Assume there are N workflows, and each workflow has M nodes. Each workflow would generate $O(M)$ subtrees. The clustering algorithm needs pairwise comparison, causing $O(M^2N^2)$ time complexity in total.

Optimization: Parallelization. The mining algorithm can be easily parallelized because the comparison of each pair is independent on that of other pairs. In parallelized mining, WRS first generates all comparison pairs and put them in a queue, and each thread iteratively fetches a pair from the queue, makes the comparison, and puts the result into the final aggregated records.

Optimization: Incrementally Mining New Workflows. When a new workflow is added to the database, WRS would generate all its subtrees and compare them with existing subtrees to find similar pairs, which would be incrementally recorded as atomic structures. The time complexity is $O(M^2N)$.

TABLE II: An Example of Atomic Structure Recommendation

Workflow	Description of workflow	Atomic Structure affiliation
F	Keyword 1,2,3	Structure A, B, C
G	Keyword 1,3	Structure A, B
H	Keyword 1,2	Structure A, C
...	...	...
X	Description of new workflow	A? B? C?

C. Organizing Atomic Structures

Each tree would output one or more atomic structures, and the total number of atomic structures could be large. Storing the atomic structures linearly is not friendly for frequent retrieval. Thus, we build two data structures to organize the atomic structures.

Atomic Structures Hierarchy. In a naïve design, all atomic structures are stored sequentially in the database. And when an operator inputs a (partial) workflow to find a similar structure, all atomic structures need to be compared with the query sequentially to find and return the closest ones. This sequential comparison is substantially time-consuming, violating the requirement of fast response in a recommendation system.

WRS organizes atomic structures in a two-level hierarchy, which can accelerate the retrieval process. WRS first clusters atomic structures into a few classes,[1] and choose one structure within a class that has the minimum overall distance to all other structures in the same class. We name that structure as the "representative structure" of the class. Then all atomic structures are organized with the "representative structures" in the first level and all structures in the same class as their children in the second level. (Fig.8)

During the atomic structure retrieval, the input workflow is compared with the representative structures first with a looser threshold K to identify its class, and then compared with all atomic structures within the class to identify and return the closest ones.

Keyword Indexing for Atomic Structure. WRS associates the keywords in a rule's description with its atomic structures, which supports the query interface with keywords. First, WRS computes TF-IDF[40] of each word in all rules' descriptions. And the words with a high score are selected as keywords.

And then, for each rule, WRS associates its keywords with its atomic structures as tuples using product. For the example in TABLE II, workflow H would create tuples of <1, A>, <1, C>, <2, A>, and <2, C>.

Thirdly, all tuples of all rules are put together and compressed as 3-tuples <keyword, atomic_struct, count>, where the count is the number of the appearance of <keyword,

[1]DBSCAN computes pairwise distance and iteratively merges nodes who are close enough into one class. It has a parameter to decide whether two data points are "close" enough, but it does not need to set the number of clusters in advance.

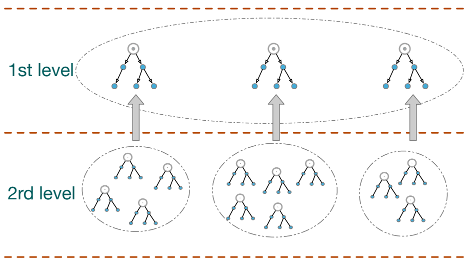


Fig. 8: Hierarchical Pattern Retrieval

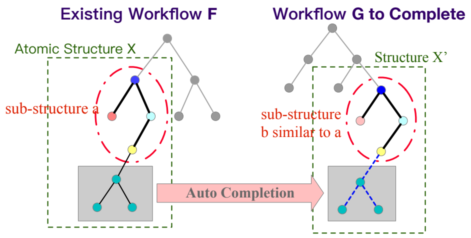
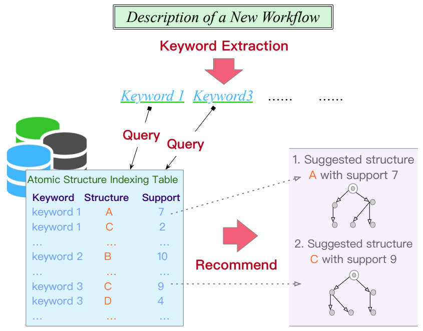


Fig. 9: Workflow Auto Completion


Description of a New Workflow
Keyword Extraction
Keyword 1
Keyword3
......
......
Query
Query
Recommend
1. Suggested structure A with support 7
2. Suggested structure C with support 9

Atomic Structure Indexing Table

Keyword	Structure	Support
keyword 1	A	7
keyword 1	C	2
...	...	...
keyword 2	B	10
...	...	...
keyword 3	C	9
keyword 3	D	4
...	...	...

Fig. 10: Structure Recommendation

atomic_struct> among all tuples. The count is also called the "support degree" of the pair of keywords and atomic_struct.

Finally, the searching index is built on keywords to find each tuple. In the actual implementation, we sort 3-tuples first by keywords (alphabet) and then by support degree (descending order). The example in TABLE II is summarized as the index in Fig. 10, e.g., <keyword 1, struct A, 7>.

When the query interface `StructIndex()` is called, the input word is searched in the indexing system to find the matching keywords, and the top few tuples (its workflow) of the keyword are returned.

D. Two Applications

WRS supports two applications to aid operators in building remediation rules — workflow auto completion and workflow recommendation.

Workflow Auto Completion. When an operator is in the progress of writing a workflow, WRS would detect the current node (i.e., the node has the cursor focus or the last edited node). All the subtrees containing the current node are fetched and used to search in the retrieval system.

First, for each subtree in the current workflow, WRS compares it with all level-1 representative atomic structures. Since the current workflow (as well as its subtrees) is not complete, when comparing a subtree (assuming size N) with an atomic structure, WRS first enumerates all subtrees of the atomic structure that contains the root node and is of size N; and WRS then compares the current workflow's subtrees with the atomic structure's subtrees to find the similar structures (smaller than the threshold K in §III-B).

Second, once similar level-1 atomic structures are found, WRS further repeats the same process in its level-2 clusters. Finally, the atomic structures which contains subtrees that are closest to the workflow's ones are returned to the operator as references. (Fig. 9)

Atomic Structure Recommendation. With the atomic structure indexing system, when an operator is creating a remediation rule and writing the description, WRS would use each word in the description to search the indexing system and return the atomic structures whose keywords are contained in the new rule's description. The returned atomic structures are displayed in descending order by their support degrees. (Fig. 10)

TABLE III: Dataset Sources and Usage in Experiments

Dataset	Number of Workflows	Experiment
Model 1	115	Training Set (76.6%)
Model 2	226	
Model 3	184	
Model 4	161	
Model 5	180	
Model 6	265	Test Set (23.4%)

IV. Implementation

We implement WRS in Java. The TED algorithm is adopted from APTED[35], [36], and we customize the similarity computation part. The semantic similarity measurement algorithm is implemented based on word2vec[20]. The remediation rules, including the workflows, are implemented in the format of JSON. Other parts in WRS, including atomic structure mining, hierarchical retrieval, and keyword indexing, are implemented in JAVA, which contains around 1500 lines of code.

Extra Refinement in Deployment. In production network, we integrate WRS with a remediation system built on StackStorm[8]. WRS is a computer-aided system, which gives recommendations to the operator. It may provide false positives, i.e., recommending an atomic structure that is not useful. So we also allow the operator to mark a recommended atomic structure as "unuseful", like thumb up or thumb down in social networks. "Unuseful" structures would be put to a lower rank in the future recommendation.

V. Evaluation

A. Experiment Settings

Dataset. We formalize the remediation rules from maintenance manual of switches or routers, which are manufactured by a device vendor and prevalently deployed in the cloud. The size of the dataset is shown in TABLE III. There are 1131 workflows in total, and we separate them into two classes in the evaluation — all workflows from Model 1-5 (76.6%) are used for training, and those from Model 6 (23.4%) are used for the test.

Metrics. In all experiments, we measure the execution time to evaluate the algorithm's efficiency. In atomic structure mining, we ask the cloud operators to validate whether the algorithm output is truly an atomic structure and define the precision as the ratio of operator-validated ones over all the algorithm output ones. In the two applications, we use the training set to build WRS's database and use the two

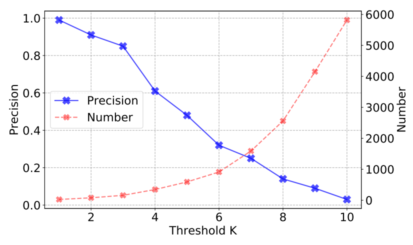


Fig. 11: Number and Precision of Atomic Structures, Varying the Threshold

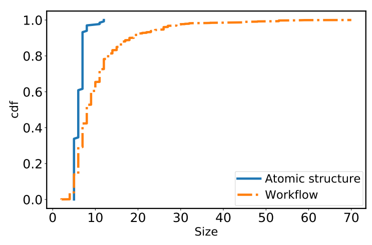


Fig. 12: CDF of the Size of Workflows and the Atomic Structures

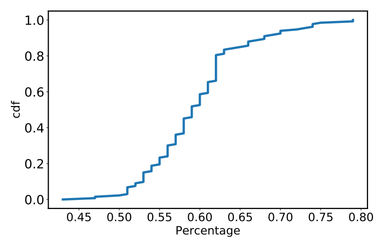


Fig. 13: Percentage of Nodes in Atomic Structure over Nodes in Workflow

applications to recommend structures for Model 6 (the test set). The TED distance between the recommendation and the ground truth (in the test set of Model 6) measures the accuracy.

Environment. All experiments are conducted on a workstation with 3.1 GHz Quad-Core Intel Core i7 and 16 GB memory.

Parameter Tuning. The similarity threshold to identify whether two trees are similar is a key parameter, i.e., K in §III-B. Fig. 11 shows the number and the precision of atomic structures that are mined with various threshold. We can observe that a larger threshold would lead to more atomic structures to be found, but the precision (that they are really atomic structures) would be less. We observe that when K is 3, the number of workflows increases rapidly to a moderately high value (158) and the precision is kept at a reasonably high level (0.85). Thus, we choose K empirically to be 3 in our experiments.

B. Offline Atomic Structure Mining

Results. The mining algorithm outputs 134 atomic structures with the dataset, which covers 34%, 32%, 20%, 22%, 29% of workflows for device Model 1 to 5. This result validates the existence of representative atomic structures in the operation of devices of the same type. While the coverage is not 100%, it can still save manual workload for operators. With larger datasets, the coverage can be improved.

Fig.12 shows the CDF of the size of workflows and the atomic structures. The workflow follows a long-tail distribution. The size of workflows varies from 6 to 70 nodes with a median size 8, and that of atomic structures from 5 to 12 with a median of 6.

For each workflow with atomic structure, we compute the percentage of the nodes in atomic structures among the whole workflow's nodes. And we draw the CDF of the percentage in Fig.13. The median value is 59%, which is the workload that WRS can possibly be saved for the operator by using WRS.

Execution Time. We vary the number of workflows selected from the training set in the atomic structure mining and show the execution time in Fig. 14 and per-flow runtime of incremental mining in Fig. 15. We have the following observations. First, the mining time increases quadratically with the number of workflows, which matches the theoretically time complexity $O(M^2N^2)$. Second, with the parallelization of 4 CPU cores, the computation time is reduced to 1/4 of the single thread mode. Third, the time to incrementally mine a workflow is proportional to the number of existing workflows. Considering that the mining process is offline, the execution time is practical for WRS to be applied the first time to a production system — 74 seconds for a remediation database with 226 rules.

C. Two Online Applications

The experiments in this subsection are conducted on the mining results of the full training set.

Accuracy. The two applications would recommend candidates to the operators. Thus, we use "top-N" recommendation list to evaluate its accuracy. In the evaluation of auto completion, we choose workflows with atomic structures from the test set and then remove 1/3 nodes of the workflows. We let the auto completion application recommend top-N atomic structures for the existing part; if one structure in the recommended list is similar (within the similarity threshold K), this recommendation is considered as "accurate". In the evaluation of atomic structure recommendation, for each workflow in the test set, the application would use its description to recommend a list of atomic structures, and the "accuracy" is similarly defined as that for the auto completion.

Fig.16 shows the accuracy in both applications with varying N from 1 to 10. We do not recommend more than ten atomic structures to operators because that would cause extra complexity for the operator to make choices. We have the following observations.

First, it is obvious that recommending more atomic structures (larger N) would be more likely to contain a useful one in the list (more accurate). Second, the accuracy of both applications increases with the length of the recommendation list, and the accuracy is in a range that makes both applications applicable to the production, i.e., 64% to 95% for auto completion and 35% to 77% for recommendation. Third, in keyword-based recommendation, the accuracy shows a slight marginal increase when the N is larger than 5, and its ultimate accuracy is 75%. Although this accuracy is acceptable for practical deployment, there are still approaches to improve it — the quality of keyword extraction (TF-IDF) depends closely on the corpus. We encourage the operator to clarify and specify the description of remediation rules to make it concise and representative.

Execution Time. In auto completion, the algorithm needs to search all atomic structures. It takes 15.6s to sequentially search all atomic structures when the number is set to 120, as Fig. 17 shows. The hierarchical retrieval system could

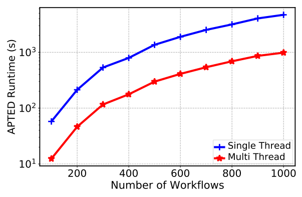


Fig. 14: Runtime of Structure Mining, Tuning the Number of Workflows

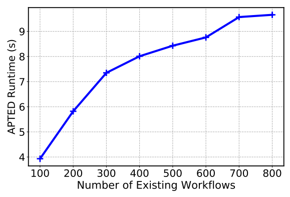


Fig. 15: Runtime Using Incremental Mining, Tuning the Number of Existing Workflows

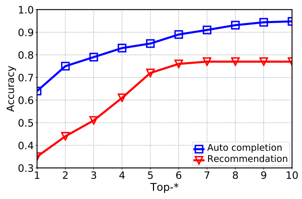


Fig. 16: Accuracy of Auto Completion and Atomic Structure Recommendation

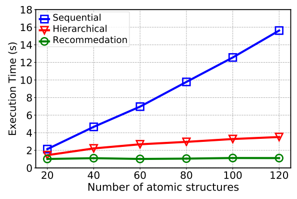


Fig. 17: Runtime of Auto Completion and Atomic Structure Recommendation

significantly reduce the search time to 3.5s. Fig. 17 also shows the execution time of sequential search and hierarchical search with varying atomic structures. The sequential search time increases linearly with the number of atomic structures, and the hierarchical retrieval increases sub-linearly (approximately to be $O(N^{0.5})$).

For keyword-based recommendation, the execution time to make a recommendation is on average 0.9s for production data and 1.1s for synthesis data in Fig.17. It makes little difference since we have indexed the recommendations with keywords.

In conclusion, compared to 10X minutes (in our survey) used to build a new workflow manually, it is efficient to take several seconds to complete or recommend atomic structures with high accuracy.

D. Progress In Production

Currently, we deploy WRS in a cloud provider with 6 data centers and tens of thousands of machines. In the years from 2018 to 2020, WRS supports building remediation rules, whose number increases more than 15x from 23 initiatives.

VI. Related Work

Remediation Systems. Remediation systems are introduced into modern information infrastructure operation, e.g., FBAR[10] in Facebook, Naoru[31] in Dropbox, Winston[7] in NetFlix, Azure Automation[1] in Microsoft, Mistral[2] in OpenStack, and StackStorm[8]. They follow the architecture in §II and focus on the compatibility with their target system in operation. WRS not only provides the platform-independent data representation but also recommends rules to populate the remediation database.

Methodologies in Remediation. There are other methodologies to systematically remedy a system. For example, Operator pattern[6] in Kubernetes uses low-level codes to defined the operation workflow and further enable automation. Trigger-action programming is popular in IoT fields to achieve automation. [18], [21], [44] WRS's abstraction is inspired by the workflow in the maintenance book, which is more operator readable.

A class of literature describes the fault localization or root cause analysis in a specific system or scenario. [26], [24], [43], [14], [46] use active probing or network traffic analysis to determine the issues. These proactive methods can be scheduled as workflows and deployed in remediation systems. Therefore, these methods would directly benefit from WRS.

Other kind of methods are passive since they only monitor the network without interactions and carry out analysis based on collected data. [15], [22], [13], [39], [45], [27], [19], [38], [25], [42] apply machine-learning methods and natural language processing to identify the root cause from historical KPIs and system logs. Some other works[29], [33], [16] mine event correlations using alarms and logs data to locate the root cause. These methods might perform better in certain sceneries or datasets and can serve the remediation collaboratively with WRS.

The remediation system is more suitable to incorporate the ones with dependency graph, causality analysis, and diagnostic rules, but does not suit learning-based well. Therefore, WRS can help to build the diagnostic rules for the former solutions.

Tree Edit Distance. There are other tree edit algorithms, e.g., TopDiff[37] and approximation[17]. Building SS-TED in WRS based on them can improve the efficiency of the atomic structure mining, which we leave as the future optimization to WRS.

Semantic Similarity. Using word embedding to map a word or sentence to a vector space and computing their distance is a typical method to get the semantic similarity. There are other options such as WMD[30] and supervised WMD[28] to improve the accuracy of the similarity.

Datasets for Operation Research. [4], [5], [12] are typical datasets for infrastructure operation research. But they are not in the format to describe a "workflow" (e.g., time series in [4], logs in [5], [12], tickets in [3]). WRS abstracts the operation process as a tree structure, which is widely used in operation (e.g., maintenance manual), and we release the dataset to the academia for research.

VII. Conclusion

We built WRS, which is a recommendation system that helps operators when they are building remediation rules. WRS recommends structures from existing rules to operators for the creation of new rules. WRS uses customized TED algorithm to extract atomic structures in existing workflows and organizes the atomic structures in a two-level hierarchy and keyword indexing format. We built an auto completion application based on the two-level hierarchy and a workflow recommendation application based on keyword indexing. Our implementation and evaluation shows that WRS's approach of extracting and organizing atomic structures in workflows can significantly improve the accuracy and execution time of workflow recommendation.

REFERENCES

[1] https://azure.microsoft.com/en-us/services/automation/.

[2] https://docs.openstack.org/mistral/latest/.

[3] https://github.com/karolzak/support-tickets-classification.

[4] https://github.com/khundman/telemanom.

[5] https://github.com/logpai/loghub.

[6] https://kubernetes.io/docs/concepts/extend-kubernetes/operator/.

[7] https://netflixtechblog.com/introducing-winston-event-driven-diagnostic-and-remediation-platform-46ce39aa81cc.

[8] https://stackstorm.com.

[9] https://support.huawei.com/enterprise/en/index.html.

[10] https://www.facebook.com/notes/facebook-engineering/making-facebook-self-healing/10150275248698920.

[11] https://www.rundeck.com/open-source.

[12] https://www.usenix.org/cfdr.

[13] Bhavish Aggarwal, Ranjita Bhagwan, Tathagata Das, Siddharth Eswaran, Venkata N. Padmanabhan, and Geoffrey M. Voelker. Netprints: Diagnosing home network misconfigurations using shared knowledge. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, page 349–364, USA, 2009. USENIX Association.

[14] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, Renton, WA, April 2018. USENIX Association.

[15] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 440–453, New York, NY, USA, 2016. Association for Computing Machinery.

[16] Ranjita Bhagwan, Rahul Kumar, Ramachandran Ramjee, George Varghese, Surjyakanta Mohapatra, Hemanth Manoharan, and Piyush Shah. Adtributor: Revenue debugging in advertising systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 43–55, Seattle, WA, April 2014. USENIX Association.

[17] Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. 1+epsilon approximation of tree edit distance in quadratic time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 709–720, New York, NY, USA, 2019. Association for Computing Machinery.

[18] Ryan Chard, Rafael Vescovi, Ming Du, Hanyu Li, Kyle Chard, Steve Tuecke, Narayanan Kasthuri, and Ian Foster. High-throughput neuroanatomy and trigger-action programming: A case study in research automation. In *Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science*, AI-Science'18, New York, NY, USA, 2018. Association for Computing Machinery.

[19] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 36–43, 2004.

[20] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[21] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. *Empowering End Users in Debugging Trigger-Action Rules*, page 1–13. Association for Computing Machinery, New York, NY, USA, 2019.

[22] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, Konstantina Papagiannaki, and Peter Steenkiste. Identifying the root cause of video streaming issues on mobile devices. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery.

[23] Aaron Gember-Jacobson, Wenfei Wu, Xiujun Li, Aditya Akella, and Ratul Mahajan. Management plane analytics. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 395–408, New York, NY, USA, 2015. Association for Computing Machinery.

[24] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 139–152, New York, NY, USA, 2015. Association for Computing Machinery.

[25] Jianglei Han, Ka Hian Goh, Aixin Sun, and Mohammad Akbari. Towards effective extraction and linking of software mentions from user-generated support tickets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 2263–2271, New York, NY, USA, 2018. Association for Computing Machinery.

[26] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 71–85, Seattle, WA, April 2014. USENIX Association.

[27] Jiyao Hu, Zhenyu Zhou, Xiaowei Yang, Jacob Malone, and Jonathan W Williams. Cablemon: Improving the reliability of cable broadband networks via proactive network maintenance. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 619–632, Santa Clara, CA, February 2020. USENIX Association.

[28] Gao Huang, Chuan Quo, Matt J Kusner, Yu Sun, Kilian Q Weinberger, and Fei Sha. Supervised word mover's distance. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4869–4877, 2016.

[29] S. Kobayashi, K. Otomo, K. Fukuda, and H. Esaki. Mining causality of network events in log data. *IEEE Transactions on Network and Service Management*, 15(1):53–67, 2018.

[30] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966. PMLR, 2015.

[31] David Mah. Bridging the safety gap from scripts to full auto-remediation. Dublin, July 2016. USENIX Association.

[32] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. A large scale study of data center network reliability. In *Proceedings of the Internet Measurement Conference 2018*, pages 393–407. ACM, 2018.

[33] X. Nie, Y. Zhao, K. Sui, D. Pei, Y. Chen, and X. Qu. Mining causality graph for automatic web-based service diagnosis. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2016.

[34] Mateusz Pawlik and Nikolaus Augsten. Rted: A robust algorithm for the tree edit distance. *Proc. VLDB Endow.*, 5(4):334–345, December 2011.

[35] Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)*, 40(1):3, 2015.

[36] Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157 – 173, 2016.

[37] Mateusz Pawlik and Nikolaus Augsten. Minimal edit-based diffs for large trees. In *Proceedings of the 29th ACM International Conference on Information &amp; Knowledge Management*, CIKM '20, page 1225–1234, New York, NY, USA, 2020. Association for Computing Machinery.

[38] Aidi Pi, Wei Chen, Shaoqi Wang, and Xiaobo Zhou. Semantic-aware workflow construction and analysis for distributed data analytics systems. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 255–266, New York, NY, USA, 2019. Association for Computing Machinery.

[39] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 127–141, 2013.

[40] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[41] Joseph Severini, Radhika Niranjan Mysore, Vyas Sekar, Sujata Banerjee, and Michael K. Reiter. The netivus manifesto: Making collaborative network management easier for the rest of us. *SIGCOMM Comput. Commun. Rev.*, 51(2):10–17, May 2021.

[42] Vikrant Shimpi, Maitreya Natu, Vaishali Sadaphal, and Vaishali Kulkarni. Problem identification by mining trouble tickets. In *Proceedings of the 20th International Conference on Management of Data*, pages 76–86, 2014.

[43] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. Netbouncer: Active device and link failure localization in data center networks. In *16th USENIX*

Symposium on Networked Systems Design and Implementation (NSDI 19), pages 599–614, Boston, MA, February 2019. USENIX Association.

[44] Valerie Zhao, Lefan Zhang, Bo Wang, Michael L. Littman, Shan Lu, and Blase Ur. *Understanding Trigger-Action Programs Through Novel Visualizations of Program Differences*. Association for Computing Machinery, New York, NY, USA, 2021.

[45] Wubai Zhou, Wei Xue, Ramesh Baral, Qing Wang, Chunqiu Zeng, Tao Li, Jian Xu, Zheng Liu, Larisa Shwartz, and Genady Ya. Grabarnik. Star: A system for ticket analysis and resolution. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 2181–2190, New York, NY, USA, 2017. Association for Computing Machinery.

[46] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 479–491, 2015.