A Framework For Misconfiguration Diagnosis in Interconnected Multiparty Systems

Malek Athamnah, Amitangshu Pal and Krishna Kant
Computer and Information Sciences, Temple University, Philadelphia, PA 19122
E-mail:{mathamna,amitangshu.pal,kkant}@temple.edu

***Abstract*—Most large systems involve multiple zones with distinct ownership or control and one or more such zones must be traversed by the transactions. Therefore, in case of misconfigurations, it is necessary to conduct tests that go across parties. Such tests are complex as they must consider composition of test functionalities provided by each party and the feasible tests must abide by the access restrictions. In this paper we propose a framework for defining test functionalities their composition, and their access control. We then discuss an efficient algorithm to determine the realization of the given test via valid compositions of individual functionalities in a way to minimize the number of parties involved.**

***Keywords:* Misconfiguration, multiparty system, access control, testing, diagnosis.**

I. Introduction

With increasing societal reliance on and complexity and diversity of automated services, the service outages and slowdowns become a major threat to business continuity and goodwill. A recent study of best practices in the Fortune 1000 companies [1] has shown that the average total cost of unplanned application downtime per year is $1.25 billion to $2.5 billion overall, and on the average $0.5 million to $1 million per hour for critical application failures in large organizations.

It has been well recognized that inconsistent changes relating to configurations and IT production environments are the top root causes of system outages or performance problems [2], [3], [4]. Past studies have indicated that up to 80% of downtimes of mission-critical applications are caused by mistakes, miscommunications or misunderstanding related to changes [5] and up to 85% of performance incidents can be traced to changes [6]. However, the root cause identification of performance problems usually takes one week on average [6], and often much longer due to misleading messages [5]. The ongoing "DevOps" transformation of IT, which melds the line between development and operations and thereby expects to provide *continuous deployment* [7], will further exacerbate the problem of misconfiguration related hiccups, unless effective solutions are found to quickly locate the root causes of the problem and fix them.

Large-scale cyber and cyberphysical infrastructures are naturally composed of multiple administrative domains, each managed or controlled by a party with potentially unique policies and configurations that are not shared with other parties. Yet user requests pass through the infrastructure of multiple parties. For example, in the typical situation of a customer in an organization accessing a remote web service, at least 3 parties are involved: the local organization, the internet service provider (ISP) that provides wide area network access, and the data center that hosts the web service. Each of these parties has its own configuration and test procedures to diagnose problems. However, in order to properly diagnose end to end problem, it is essential to run multiparty tests, and these would require cross-party access. We envision them as compositions of test probes provided by individual parties along with suitable cross party access rules. These access rules may not be statically granted, and instead granted specifically for testing purposes.

The multiparty access problem arises in a variety of scenarios, even including those involving a single enterprise. For example, large enterprises often have multiple business that do not freely share information and thus may use separate private clouds hosted in the same server farm. Even within an owned data center, there are often different teams with different expertise and responsibilities, (e.g., network team, security team, and database team), each of them has their own configuration and perhaps some limited access by other teams. In all these cases, when problems arise, a multi-party testing may be essential and such testing may grant more access rights than in the normal situation. In any case, given a basic test functionalities and access rights, one could ask several questions such as: Is the set of given access rights adequate to do a multiparty test? If a test can be carried out in multiple ways, what is the shortest test, or test involving fewest parties? These are the types of questions that we are interested in addressing in this paper.

It is important to point out the issues that we do not address in this paper. First, we assume that the parties involved have a collaborative, rather than adversarial relationship. In other words, the parties do not attempt to circumvent the specified access rules, do not corrupt/alter the data provided to other parties, and do not hold back or filter any legitimate information. We assume that all collaborating parties are mutually authenticated using standard mechanisms, and the global controller tracks and limits the rate and number of concurrent tests in order to prevent DDoS scenarios.

Accessing data across parties can be technically challenging

This research was supported by the NSF grant CNS-1527346.

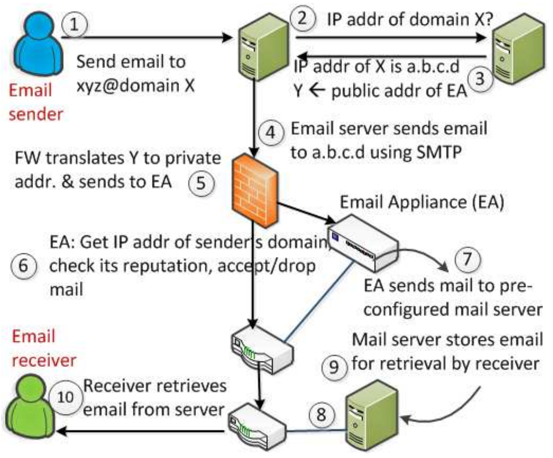

1
Send email to xyz@domain X
Email sender
2 IP addr of domain X?
IP addr of X is a.b.c.d
Y ← public addr of EA 3
4 Email server sends email to a.b.c.d using SMTP
FW translates Y to private addr. & sends to EA 5
Email Appliance (EA)
6 EA: Get IP addr of sender's domain, check its reputation, accept/drop mail
7 EA sends mail to pre-configured mail server
Email receiver
Mail server stores email 9 for retrieval by receiver
10 Receiver retrieves email from server
8

Fig. 1: Typical Email Flow

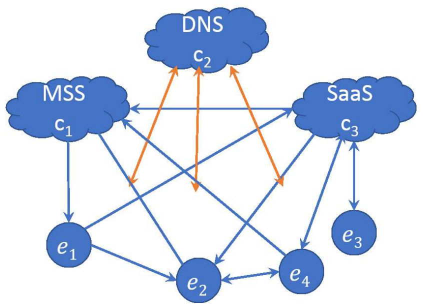


Fig. 2: Email Configurations

due to varying format and semantics of the data used by them, but these are not the focus of this work. Thus we proceed with the simple assumption that each party creates a "stub" for its data for uniform access by others.

Building a comprehensive misconfiguration diagnosis infrastructure involves many significant research, design, and implementation challenges. These include (a) exploring constraints to be exploited by the probes for diagnosis, (b) defining and handling access control across parties, (c) test design and selection, and overall testing infrastructure design, implementation, and evaluation. We note that this paper is not concerned with the design of the probes provided by various parties or the design of suitable tests to uncover a specific issue. Instead our focus is on whether the given test can be implemented using the given probes, and if so, how it can be implemented most efficiently.

The outline of the paper is as follows. Section II provides an example of a common multiparty system, namely the email system. Section III discusses various issues in building multiparty tests from individual probes provided by the parties. Section IV defines the problem of test selection formally and proposes a solution. Section V then shows sample performance evaluation. Related works are discussed in section VI. Finally, section VII concludes the discussion.

II. An Example of Multiparty Service

In this section we discuss in some depth the configuration of Internet email, the inherent involvement of multiple parties, and how multiparty testing would be needed to diagnose the misconfiguration.

Fig. 1 shows the typical flow for email involving the email filtering appliance (EA). The E-mail server resolves the E-mail domain name to the public IP address of the domain using the DNS service. It then sends the E-mail using SMTP (Simple Mail Transfer Protocol) to the corresponding IP address. The email is intercepted by the enterprise firewall (FW), which translates the public IP address of EA to its (private) DMZ IP address and forwards traffic to the EA. The EA then does a DNS query on the sender domain name, compares the IP address of the sender to its own SensorBase database and determines the reputation score of the sender. It rejects the E-mail if it falls within a pre-configured reputation score, and sends a failure email back to the sender. The receiver then retrieves the email at its convenience.

The key resources here are obviously DNS, FW, EA, and Mail Server (MS). The DNS can be probed directly, but others generally do not provide any direct interface for querying. Thus, it is necessary to design and install Resource Probing Module (RPM) on others. With that, and assuming that FW, EA and MS are owned by the same party, it is adequate to have a single Mirror for this email system. Unfortunately, the email setups are rarely this simple, as illustrated in Fig. 2, which shows 3 popular configuration methods and highlights the need for multi-party coordination.

In Fig. 2, e_i's denote different enterprises, and c_j's denote different cloud or service providers. Enterprise e_1 delegates email filtering to a third party c_1, a Managed Security Services (MSS) provider. In this case, any incoming email to e_1 is delivered through cloud c_1. Note that the outbound mail from e_1 is configured to be sent directly as shown by arrows to e_2 and c_3. (Not showing all arrows to avoid clutter.) In contrast, e_2 and e_4 configure their email service locally for direct sends/receives to/from other entities. Finally, e_3 uses a cloud SaaS service for email; i.e., both sends and receives happen through the provider c_3. This means that no email server talks with e_3 directly and is instead forced to go through c_3. In all cases, the enterprises use a public DNS service hosted by the service provider c_2.

In order to diagnose email problems in such an environment, each party will need to setup a number of probes relating to various problems in its sphere of control. Some common email issues include the following: (a) Spammers could use email servers to relay emails through them, usually by pretending to be the higher distance MX servers for the domain that may be engaged under heavy load or failure of the primary. (b) When an email DNS server has an "A" record for the sender but not the reverse record, the recipient email server may force all emails into the junk/spam folder or reject them outright. The same happens if the sender erroneously gets on a blacklist

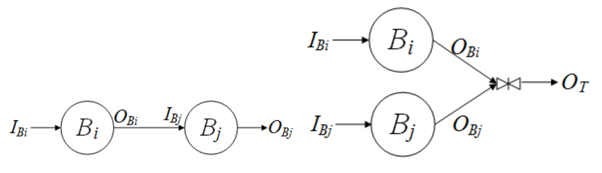


Fig. 3: Combination of probes in (a) series, and (b) parallel.

(not uncommon). (c) Email could be disrupted from other misconfigurations such as those in the ISP network or those in the email appliances (EA) discussed above.

III. Diagnosis in Multiparty Systems

Diagnosis of problems in a multi party environment is particularly difficult because the infrastructure owned or accessible by each party is differently configured/managed and other parties have no visibility into or understanding of a party's infrastructure, configuration, or compatibility issues across parties. The multiparty diagnosis problem can be eased by each party providing a set of "probes" with well defined interfaces and cross-party access rights to the probes so that a multiparty test can be constructed out of such probes. Note that several parties may provide the same probing capability; this is quite normal since the available probing capability often depends on the location and vantage point of the party. In fact, a large organization often has multiple physical locations, each with different access capabilities; for example, a branch office may not have the same visibility and probing capability as the main office. For the purposes of this paper, we shall consider such multiple locations as different "parties" with the access rules across our parties reflecting any organizations boundary considerations. In particular, if an organization has access to some probing capabilities, they will likely be available from multiple locations of the organization (which we are considering as different "parties" in our modeling).

The key to such multi-organizational and multi-location testing mechanism is a clear definition of semantics and format of the inputs and outputs of each probe such that they can be meaningfully connected across parties. This involves several issues that we discuss in the following, namely, how to build multiparty tests from party-specific probes, how to define cross-party access rights, and various questions that would interesting to study regarding the testing.

A. Generating Tests from Probes

Consider parties P_i, and P_j, each of which define the probe B_i and B_j respectively. Let (I_{B_i}, O_{B_i}) and (I_{B_j}, O_{B_j}) denote the input/output of probes B_i and B_j respectively. Now consider a *requesting party* P_R that constructs a multiparty test (MPT) T out of these probes and runs it, eventually receiving the results.

The two obvious ways of combining the probes are: (a) **Series**, i.e., run B_i and use O_{B_i} to define I_{B_j}. Then run B_j and use O_{B_j} to define the test output O_T for P_R, and (b) **Parallel**, i.e., run B_i and B_j concurrently using I_{B_i}, O_{B_i} and use a some composition O_{B_i}, O_{B_j} to generate the test output O_T for P_R. This is depicted in Fig. 3. In the series case, I_{B_j} needs to be compatible with O_{B_i}, which means that (a) either both probes B_i and B_j are simple, or both are structured, and (b) it is possible to derive I_{B_j} from O_{B_i}.

For a parallel test, the crucial part is the "join" node, which represents some type of composition of the outputs (O_{B_i} and O_{B_j}) received from the preceding probes. In this case, we requires both outputs to be either simple or both structured. In case of structured outputs, this composition could be any of the standard relational operations such as equi-join, union, intersection, etc. In case of simply outputs, there is no explicit composition, both inputs are provided to the next probe.

In general, the test graph could be an arbitrary acyclic graph with a single source and single destination. However, such graphs become tricky with respect to I/O compatibility; furthermore, finding efficient ways of conducting tests with arbitrary structure becomes quite difficult. Therefore, in this paper, we assume that the test graphs are recursively composed of series or parallel subcomponents.

One other point in defining the tests is whether to specify the needed probes uniquely (e.g., via a unique id of the probe) or more generally. The constituent probes of a test are specified uniquely, the test construction problem is trivial (since there is no choice). A somewhat more general method is to specify probe in terms of its input, functionality and output (simply called "functionality" for short), rather than a unique id. This allows consideration of probes that have identical or superset of the requested functionality owned by various parties, and we will consider this as discussed later. It is possible to make the specification even more abstract, but we do not consider it here.

B. Defining Cross-Party Access Rights

The key attribute of the multiparty environment is the restrictions on cross party accesses. The access control can be specified at two levels: (a) access to the data, i.e., access to the output of a probe (or set of probes), and (b) access to the entire probe, meaning ability to give input to the probe, command its execution, and collect the output.[1] It is important to make data vs. probe access since it is possible to provide access to the output of a probe result to a party, without that party having the ability to actually run the probe. Also note that in case of a parallel arrangement of probes, the input to the next probe is derived from the composition of the outputs, only the access to this composition is crucial.

Several models are possible for connecting multiple probes with regard to access control, ranging from very strict to promiscuous. In a strict model, a party must have direct access to all the probes used in the test and also to their outputs. This

[1]Bundling probe and data access here is just for clarity in discussion; the detailed specification in section IV-A actually defines the data and probe access functionalities separately.

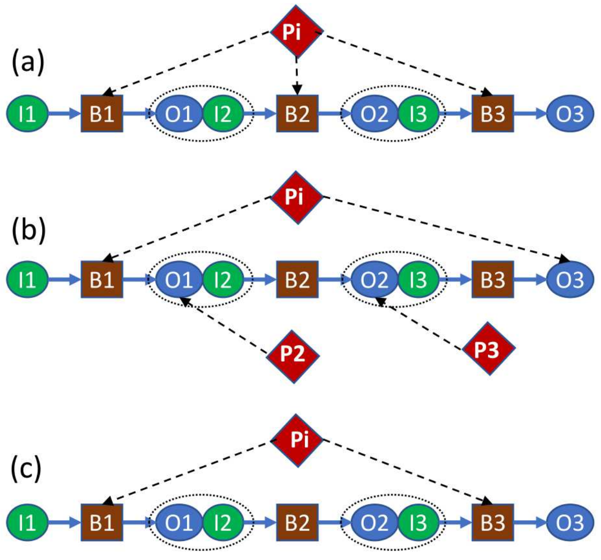


Fig. 4: Illustration of Access Possibilities

is illustrated in Fig. 4(a) for a simple test with 3 probes (B1, B2, B3) in series, but similar situation applies in general. In this figure, we have an implicit assumption that probes B1, B2, B3 are owned by parties P1, P2, and P3 respectively, and the test is to be invoked by some other party P_i. the dotted ovals show compatibility between inputs and outputs, and this is always required. More important, the figure shows that the requesting party P_i must have access to all 3 probes and their inputs and outputs.

A considerably looser model would allow the requester to only possess end point access rights, as illustrated in Fig. 4(b). Here P_i only needs the ability to run probe B1 and access the output $O3$. However, to move the access along, we need P2 to have access to the output O1 and P3 should have access to O2. With this, the entire chain is completed, which means that the test can be run and resulted obtained by P_i.

Finally, Fig. 4(c) shows the loosest model. Here P_i needs the ability to run B1 and collect result O3 (as in case (b)); however, no other intermediate access restrictions are required to complete the test.

Many other possibilities also exist, such as granting access rights based on the results of the tests; however, these 3 cases illustrate the tradeoff between flexibility of testing and the protection/accessibility of the probes. Case (a) is easy to handle in determining the feasibility and minimality of the test; however, the restrictions it places are perhaps identical to access restrictions during normal functioning. We believe that when problems arise, a looser access control model is more appropriate in order to grant the necessary accesses quickly. Case (c) is the easiest in this regard; however, it allows for many possibilities, which means that the question of minimal test construction becomes more interesting, and is in fact NP-hard, as discussed later.

C. Testing Issues

Now consider a requesting party P_R that has the rights to its desired inputs and outputs, and it wants to design a suitable MPT to obtain the output. Then, we can pose the following problems:

- Feasibility: Can we select the probes, probe graph, and probe I/O mappings for an MPT such that the output produced by the MPT is a superset of the output requested by P_R?
- Additional Rights: If the problem is infeasible, what is the minimum additional access rights that P_R needs in order to construct a feasible MPT?
- Effectiveness/Cost: If there are multiple ways to define an MPT that satisfies P_R's needs, how do we define a suitable measure of "effectiveness" or "Cost" that can be optimized?

The cost/effectiveness of the test can be defined in multiple ways. One simple definition, that we shall use in this paper, is the involvement of minimum number of parties. However, one may instead want to minimize the number of probes, or the overall cost/overhead of running the test.

These problems can be considered as extended versions of the issues addressed in our earlier work [8], [9], [10]. In particular, we have shown in [8] that the rule enforceability problem (similar to the feasibility problem above) is NP-hard, and highly efficient and effective minimum cost enforcement algorithms are developed in [9], along with further extensions and generalizations in [10]. Furthermore, we have considered the question of rule change and minimal extension of the rules in [11]. These approaches are applicable here as well, except for a significant additional complication that (a) the test is given as a graph, rather than a single rule to be enforced, (b) one needs to consider the input/output compatibility and corresponding access rights as well.

Because of these complications, we will not all these problems in this paper. In particular, we do not consider the problem of adding minimal additional access rights to make the testing feasible. We also do not consider different variations in access rights requirements for running tests (discussed in section III-B). Instead we only consider the most promiscuous case, where there may be many ways of satisfying the test.

IV. Problem Description and Solution Method

In this section we describe the problem of minimal cost test construction and show that the problem is quite complex and easily shown to be NP-Hard. We then describe a heuristic solution.

A. Preliminaries and notations

Assume that there is a set of parties $\mathbb{P} = \{P_1, P_2, \ldots, P_p\}$. Also assume that there are n probes $\mathbb{F} = \{F_1, F_2, \ldots, F_n\}$, each one can be provided/accessed by multiple parties. Each party P_i provides a set of probes $B_i \subseteq \mathbb{F}$. Each probe F_i can take a set of inputs I_i and produces an output O_i. For example, a probe can be a DNS lookup that takes an input type *domain names* and produces an output that is a numerical *IP address* corresponding to a computer service or device.

As each probe is accessed by multiple parties, we will have multiple, possibly overlapping probes across parties. Also, the parties can give controlled access to specific data from their provided probes to one another. The specific access of each probe is governed by an explicit set of rules. The rules are of two different types: *run access* and *data access*.

The run access $\mathbb{R}$ is described by 2-tuple, $\mathbb{R}(P_i, F_j)$, means party P_i is allowed to run probe F_j using an arbitrary valid input. The data access $\mathbb{D}$ is also described by a 2-tuple, $\mathbb{D}(P_i, O_j)$, which indicates that party P_i is allowed to access the data O_j (i.e., the output of probe F_j). The *success* function is the result of the combination of the run access and data access, which means that a party can successfully execute a probe. Thus a success function $\mathbb{S}$ can be described by 3-tuple, $\mathbb{S}(P_i, I_j, O_j)$, which means that the party P_i can successfully execute probe F_j with input I_j and obtain output O_j. This is denoted as $P_i \rightarrow F_j$.

We also have a controller C who can communicate with all parties. The controller has multiple roles. The controller figures out which probes from other parties can serve particular party's test, and what are the required input types and values for the probes. It also provides the best routing plan for specific test through all available probes. The controller can be thought of as a party that can be trusted to perform these functions correctly and reliably.

We also assume that there are some input/output compatibility relationships across the probes. That is, a probe F_j's input is compatible with F_i's output (denoted as $F_i \rightarrow F_j$), if F_j's input can be derived from F_i's output, i.e. $I_j \subseteq O_i$.

If a party P_R wants to run a test, it first sends a request to the controller C with all the required information. C then provides the plan for the test to P_R after considering the access rights across different parties along with a one-time certificate (OTC) for accessing the probes across other parties. The OTC comes with a timeout and can be used by P_R only for carrying out that specific test. In this paper we assume that given a test from P_R, the controller C finds out the test plan that involves minimum number of parties needed to be involved for conducting the test, which we define as *minimum party testing (MPT)* problem.

B. Problem Definition and Complexity of MPT

Given the access rights across the parties, and the input/output compatibility across the probes, the controller C generates a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edges. Here V consists of the set of parties and probes, i.e. $V = (\mathbb{P}, \mathbb{F})$. E consists of the access rights of the parties, and the input/output compatibility across the probes, i.e. $E = (P_i \rightarrow F_j, F_j \rightarrow F_k)\ \forall P_i \in \mathbb{P}$ and

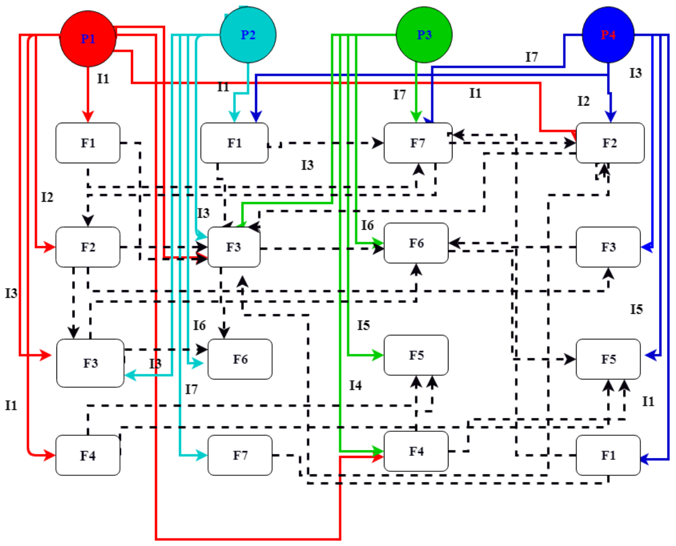


Fig. 5: Controller graph

$F_j, F_k \in \mathbb{F}$. An example of the graph G is depicted in Fig. 5. In Fig. 5, the graph G consists of four parties P_1, P_2, P_3, and P_4, each one has access to all 4 probes (shown in solid colored lines). For example, P_1 has access to $F_1 - F_4$. Furthermore, a probe can be accessed by multiple parties. For example, F_3 is accessed by P_1, P_2 and P_4. The input/output compatibility across the probes are also shown by dashed lines. For example, $F_1 \rightarrow F_3$, $F_3 \rightarrow F_6$ and so on.

A test can also be represented by a graph $\mathcal{T} = (\mathcal{T}_v, \mathcal{T}_e)$, where $\mathcal{T}_v$ and $\mathcal{T}_e$ represent a set of vertices and edges of $\mathcal{T}$. In a test graph $\mathcal{T}_v$ represents the set of probes and $\mathcal{T}_e$ represents the input/output compatibility across the probes. An example of a test graph is illustrated in Fig. 6, where $\{a, b, c, d, e\}$ is the set of probes and the edges in between them represent the input/output compatibility across the probes.

Given a test graph P_R, the controller first checks whether $\mathcal{T}$ is a legitimate test of not. The condition for checking a legitimate test is described in section IV-C. Once a test $\mathcal{T}$ is found to be legitimate, C then finds the test plan for conducting the test by solving the MPT problem. Below we first prove that the MPT problem is NP-hard. For this proof we assume a special case of the MPT problem where $\mathcal{T} \subseteq G$.

Theorem 1. *The MPT problem for given test is NP-hard.*

Proof: For this proof we reduce the well-known set cover problem to MPT. The set cover problem can be described as follows. Assume an universe $\mathbb{U}$ that consists of a set of n elements, i.e. $\mathbb{U} = \{A_1, A_2, \ldots, A_n\}$. Also assume that S is a collection of sets, i.e. $\mathbb{S} = \{S_1, S_2, \ldots, S_m\}$ where S_i is a set of elements from $\mathbb{U}$. Given the input pair $(\mathbb{U}, \mathbb{S})$ the minimum set cover problem is to find out the minimum subfamily $\mathbb{C} \subseteq \mathbb{S}$ whose union is $\mathbb{U}$.

Assume a legitimate test graph $\mathcal{T} = (\mathcal{T}_v, \mathcal{T}_e)$. Also assume that $\mathbb{F}' = \{\mathcal{T}_v \cap B_1, \mathcal{T}_v \cap B_2, \ldots, \mathcal{T}_v \cap B_p\}$ represents the

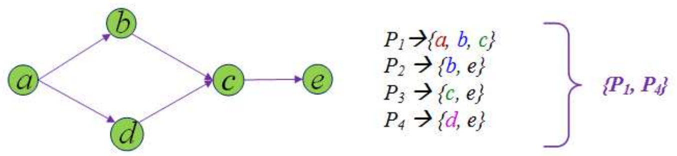


Fig. 6: Illustration of the MPT problem.

derived set of probes from original probes $\mathbb{F}$ that only includes $\mathcal{T}'s$ probes corresponding to the parties $\mathbb{P} = \{P_1, P_2, \dots, P_p\}$. Now the task is to find out the minimum $\mathbb{C}' \subseteq \mathbb{F}'$ whose union is $\mathcal{T}$. This is identical to find the MPT for conducting a given test. To map the set cover problem to MPT we construct a universe $\mathbb{U}$ which consists of the vertices of $\mathcal{T}$. We also construct $\mathbb{S}$ which is identical to $\mathbb{F}'$. Now the problem becomes how to find minimum number of sets in $\mathbb{S}$ that cover all probes in $\mathbb{U}$. In such case, if the MPT can be found in polynomial time, the set covering problem also has a polynomial solution. Thus, the MPT problem is NP-hard. ■

Algorithm 1 Greedy set cover $(\mathbb{U}, \mathbb{S})$

```
1: $\mathbb{C} = \phi$
2: while $\mathbb{U} \neq \phi$ do
3:     Pick $S_i \in \mathbb{S}$ that maximizes $|\mathbb{S} \cap \mathbb{U}|$;
4:     $\mathbb{C} = \mathbb{C} \cup S_i$;
5:     $\mathbb{U} = \mathbb{U} - S_i$;
6: end while
7: return $\mathbb{C}$;
```

Fig. 6 shows an example test graph with $\mathcal{T} \subseteq G$. Assume that P_1, P_2, P_3 and P_4 have accesses to the probes $\{a, b, c\}$, $\{b, e\}$,$\{c, e\}$ and $\{d, e\}$. Thus C finds the minimum number of sets (i.e. parties), union of which constructs the test graph $\mathcal{T}$, by solving the minimum set cover problem. For Fig. 6 we can observe that by involving P_1 and P_4, the controller can conduct $\mathcal{T}$. For solving the minimum set cover problem, we use a greedy heuristic shown in Algorithm 1. The heuristic selects the party containing the maximum number of uncovered probes at each step, until all the probes in the test graph are covered.

C. Proposed solution

In the previous subsection we assume that a test $\mathcal{T}$ is a legitimate test if $\mathcal{T} \subseteq G$. Considering the test graph of Fig. 6, b has input/output compatibility with a in G, which we denote as $(a \rightarrow b) \in G$. In this section we relax this condition to generalize the test model by using the *transitive* relations, i.e.

$$(x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge \dots \wedge (x_{n-1} \rightarrow x_n) \implies (x_1 \rightarrow x_n)$$

which we define as *transitive input/output compatibility (TIOC)*. To check whether $\mathcal{T}$ is a legitimate test or not, C checks two conditions. First it checks whether the requesting party P_R has access to the initial and terminal probes of $\mathcal{T}$, so that it can pass the initial input and access the final output of the test. (In Fig. 6, these are a and e respectively.) In addition, we also need to ensure that for each $(x \rightarrow y) \in \mathcal{T}$, there is a TIOC in between x and y in G, i.e. there exists a *path* from x to y in G. For example in Fig. 6 to ensure the test graph to be legitimate, there needs to be a path in between a to b, a to d, b to c, d to c and c to e in G.

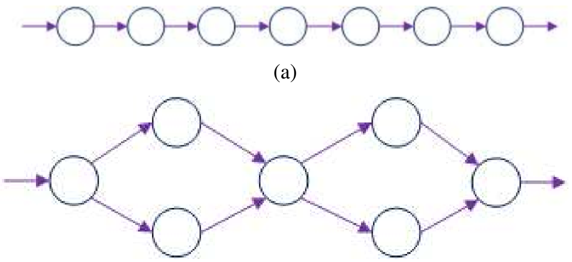


Fig. 7: Illustration of the (a) serial and (b) parallel test graphs.

If the test is legitimate, we need to find the minimum number of parties that the controller C needs to involve for conducting the test. In view of the NP-hardness result, we develop a greedy algorithm explained in Algorithm 2. For finding the minimum number of parties needed for conducting the test, the controller C first computes the K shortest paths in between the initial and terminal nodes. We use Yen's algorithm [12] to find out the K shortest paths (line 6). For each path $\Gamma_i \in \Gamma$, C records the edges of the test graph whose TIOC is satisfied by the Γ_i. It then generates all possible sets of Γ_i's whose union covers all edges of the test graph.

As an example, in Fig. 6, the union of two paths $\Gamma_1 = a \rightarrow b \rightarrow c \rightarrow e$ and $\Gamma_2 = a \rightarrow d \rightarrow c \rightarrow e$ will cover the test graph $\mathcal{T}$. Another path $\Gamma_3 = a \rightarrow b \rightarrow d \rightarrow c \rightarrow e$ will also satisfy the TIOC conditions, thus this can also be a valid set to cover $\mathcal{T}$. In reality finding all possible sets of Γ_i's for covering the test graph's edges can be exponential; therefore, we remove the paths from Γ once they are chosen by the set cover algorithm (line 20). We repeat the process until all Γ is empty or there is there is no possible combination of Γ_i's that covers the $\mathcal{T}$ (line 7-21).

For all these possible set of Γ_i's whose union covers $\mathcal{T}$, we find the minimum number of parties to run these set of probes by solving the minimum set cover problem (line 19). We then choose the set of Γ_is that can be conducted by involving the minimum number of parties (line 23).

V. Performance evaluation

For the experimental evaluation, we generate the graph G and the test graph $\mathcal{T}$ randomly. For constructing G, the access rights for parties towards the probes are generated uniformly randomly with a probability of 0.7. The input/output compatibility among the probes are also generated with a probability of 0.7. The test graphs consist of 7 distinct probes (as shown in Fig. 7) that are generated uniformly randomly among the number of available probes. Unless otherwise stated, K (the maximum number of paths) is assumed to be 50 for the experiments.

Algorithm 2 Finding the minimum number of parties MPT($\mathcal{T}, G$)

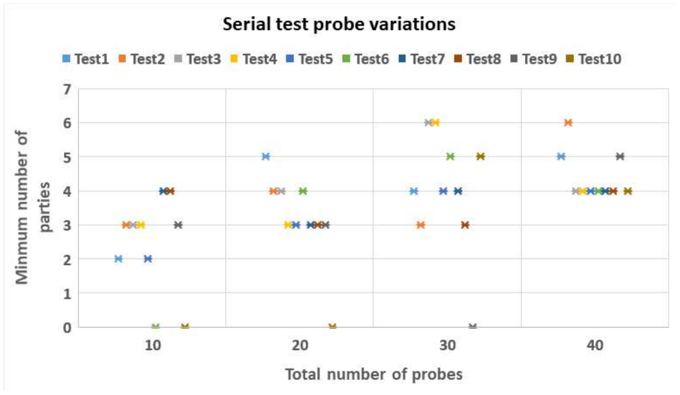
```
1:  $P_R$ : Requesting party in $\mathcal{T}$;
2:  $\mathcal{S}$ : Initial probe in $\mathcal{T}$;
3:  $\mathcal{D}$ : Terminal probe in $\mathcal{T}$;
4:  if $P_R$ has access to $\mathcal{S}$ and $\mathcal{D}$ then
5:      Min_Party = null;
6:      $\Gamma$ = Yen_Algorithm($\mathcal{S}$, $\mathcal{D}$, $G$, $K$);   ▷ Generate $K$ shortest paths
7:      while $\Gamma \neq$null && Set_Cover($\mathcal{T}$, Paths)$\neq$null do
8:          Selected_Paths = Set_Cover($\mathcal{T}$, $\Gamma$);   ▷ Cover TIOC conditions
9:          Party_subsets = NULL;
10:         for each party $P_i$ in $G$ do
11:             Party_probes = NULL
12:             for each probe $F_j$ that is accessed by $P_i$ do
13:                 if $F_j$ in Selected_Path then
14:                     Party_probes = Party_probes $\cup$ $F_j$;
15:                 end if
16:             end for
17:             Party_subsets= Party_subsets $\cup$ Party_probes;
18:         end for
19:         Min_Party=Min_Party$\cup$Set_Cover($\mathcal{T}$, Party_subsets)
20:         $\Gamma$ = $\Gamma$ - Selected_Paths;
21:     end while
22: end if
23: return the set of parties having minimum cardinality in Min_Party;
```

A. *Effect of total number of probes*

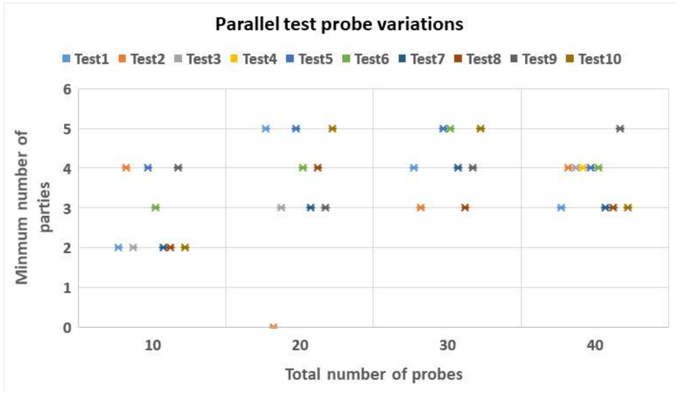
Fig. 8 shows the variation of the minimum number of parties with different number of probes. In Fig. 8 we assume 10 parties each having access to 5 probes. We randomly generate 10 different test cases and find out the minimum number of parties in each case. From Fig. 8 we can observe that the minimum number of parties almost doubles when the number of probes increases from 10 to 40. This is because as the total number of probes increases, each probe is accessed by fewer parties. This results in an increasing number of parties required to conduct the tests. Also notice that the number of parties for conducting the tests are similar for both serial and parallel cases.

B. *Effect of total number of parties*

Fig. 9 shows the variation in the number of parties required to conduct the tests with different number of parties. The total number of probes is assumed to be 15. Each party has access to 5 probes. From Fig. 9 we can observe that the number of parties for conducting the tests decreases approximately from 4 to 2 on the average when the total number of parties increase from 5 to 20. This is because as the total number of parties increases, MPT generates shorter paths between the initial and terminal probes, which results in fewer required parties for conducting the tests. The number of parties for conducting the tests does not change significantly for serial and parallel test cases.



(a)



(b)

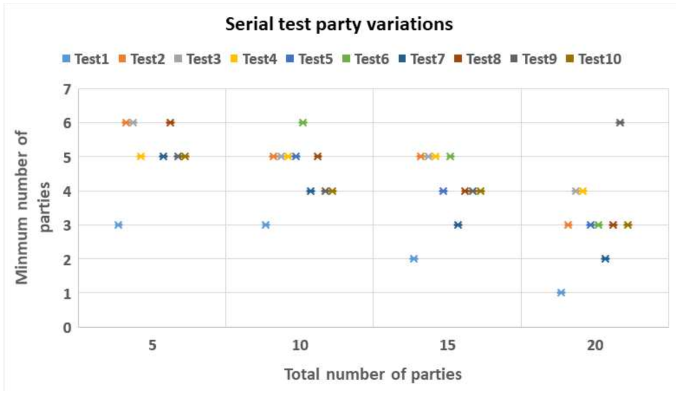
Fig. 8: Comparison of the minimum number of parties with different number of probes in G for (a) serial and (b) parallel test graphs.

C. *Effect of K*

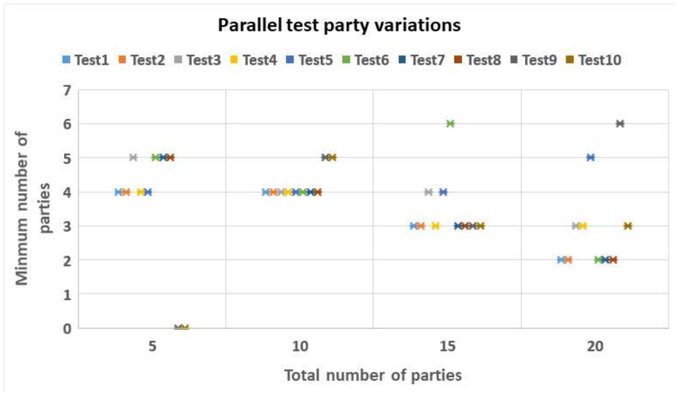
Fig. 10 shows the minimum number of parties involved in conducting two arbitrary serial and parallel test cases with increasing K. For Fig. 10 the total number of parties are assumed to be 10, each one having 5 probes. The total number of probes is assumed to be 30. From Fig. 10 we can observe that the required number of parties decreases with increasing K and then saturates beyond a certain point. This is because the MPT scheme generates more paths from the initial probe to the terminal probe as K increases, which provides more options for finding the minimum number of required parties for conducting the tests. Notice that in Fig. 10 the specific test cases for the serial and parallel are chosen arbitrarily, and thus the relative gap between the serial and parallel test cases is not important.

VI. Related Work

Change management is an active and well-studied subject in the literature and can be helpful in tracking the impact of misconfigurations and allowing for a fix. The general idea is to record the system behavior as a function of various configuration parameters as the system evolves. These records can be exploited to assess what combinations of parameters lead to “good” or “bad” behavior. Evolven [13] provides many tools based on such an approach that use machine learning



(a)



(b)

Fig. 9: Comparison of the minimum number of parties with different number of parties in G for (a) serial and (b) parallel test graphs.

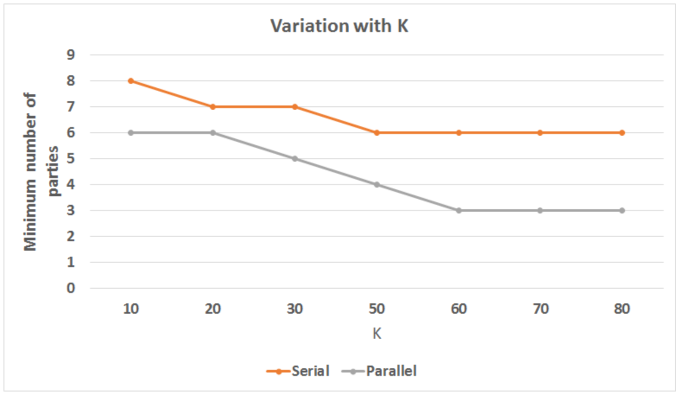


Fig. 10: Variation of number of parties required for conducting the tests with different K.

techniques to determine normal and abnormal patterns. The STRIDER tool reported in [14] applies a black box approach that assumes an underlying "statistical golden state" and uses a Hidden Markov Model based estimation procedure to detect and fix problems. Many other tools also exist such as Glean [15], PeerPressure [16], etc. However, there are no tools that tackle the multiparty environment.

Although such automated approaches can be useful in identifying and even fixing problems, they generally apply to specific types of errors and also suffer from substantial false alarms. In particular, such approaches are limited by the comprehensiveness of the collected data and the nature of dependencies. For example, if the misbehavior happens for certain combination of parameters which cannot be deduced from the available data, the approach is not useful. In short, there are still numerous instances where human involvement is essential to identify what tests might be most useful to get to the bottom of the problem. This paper concerns building an infrastructure that can automate constructing and selecting tests.

Another diagnosis approach is to define processes based on best practices such as ITIL (Information Technology Infrastructure Library), which are then automatically managed by incident management systems such as IBM's Tivoli Service Request Manager [17] and BMC Remedy Service Suite [18]. The known problems can be diagnosed automatically but others are left to administrators. Yet another method is to define a set of configurations rules which can be checked automatically. EnCore uses this approach to detect software misconfigurations [19]. This approach is simple but applies to only very specific cases.

Reference [20] proposes a REST based infrastructure to provide restricted exposure of configuration across parties. This work can be considered somewhat similar to what we are proposing. The work of [20] has discussed a configuration management infrastructure named REST in the context of a multi-domain, enterprise Web services. They have proposed a decentralized configuration change management architecture, which takes into account the distribution of configuration information and facilitates the execution of management processes across organizational boundaries while maintaining each organization's autonomy. Social media activities such as photo sharing experiences multi-party conflict scenarios too. Mechanisms for resolving such issues have been studied in [21] and could be useful for considering complex constraints.

VII. Conclusions

Misconfigurations in large scale cyber systems are known to be responsible for an overwhelming percentage of failures, poor service, and exploitation by hackers for cyber-attacks. In this paper, we explored a comprehensive, multi-party infrastructure to assist in the diagnosis by considering the access rights across different parties. We proposed an efficient way of finding out the minimum number of parties involved in conducting different multi-party test cases via valid connection of different probes (along with suitable access control) to allow for a flexible multiparty probing mechanism. We are currently building a comprehensive testing infrastructure for multi-party testing using Common Open Research Emulator (CORE) [22], [23] package. This package can support multiple, geographically separated data centers with intervening connection layers (e.g., DMZ,WAN edge, WAN, etc) and will be useful in gaining further insights into the real misconfiguration problems. We will also examine access control models other than the promiscuous model used here. Finally, we will examine minimization objectives other than the number of parties and understand how sensitive the test construction is to various types of objectives.

References

[1] S. Elliot, "Devops and the cost of downtime: Fortune 1000 best practice metrics quantified," *International Data Corporation (IDC)*, 2014.

[2] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *USENIX symposium on internet technologies and systems*, vol. 67. Seattle, WA, 2003.

[3] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 159–172. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043572

[4] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[5] F. Connolly, "Production operations – the last mile of a devops strategy," *LMC Report*, Mar 2014.

[6] W. Cappelli, "Causal analysis makes availability and performance data actionable," *Gartner Report G00273922*, Oct 2015.

[7] M. Shahin, "Architecting for devops and continuous deployment," in *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*. ACM, 2015, pp. 147–148.

[8] M. Le, K. Kant, and S. Jajodia, "Consistency and enforcement of access rules in cooperative data sharing environment," in *Computers and Security*, Nov. 2013, pp. 10–12.

[9] M. Le, K. Kant, M. Athamnah, and S. Jajodia, "Minimum cost rule enforcement for cooperative database access," *Journal of Computer Security*, vol. 24, no. 3, pp. 379–403, 2016.

[10] M. Athamnah and K. Kant, "Multiparty database sharing with generalized access rules," in *Proc. of CloudCom, Luxemburg*, Dec 2016, pp. 198–205.

[11] M. Le, K. Kant, and S. Jajodia, "Access rule consistency in cooperative data access environment," in *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom2012)*, oct. 2012, pp. 11 –20.

[12] http://www.mathworks.com/matlabcentral/fileexchange/32513-k-shortest-path-yen-s-algorithm.

[13] B. Kaluza, "Top 5 it ops challenges and how maching learning can help." [Online]. Available: https://www.evolven.com/resources.html

[14] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang, "Strider: A black-box, state-based approach to change and configuration management and support," *Science of Computer Programming*, vol. 53, no. 2, pp. 143–164, 2004.

[15] E. Kiciman and Y.-M. Wang, "Discovering correctness constraints for self-management of system configuration," in *Autonomic Computing, 2004. Proceedings. International Conference on*. IEEE, 2004, pp. 28–35.

[16] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure." in *OSDI*, vol. 4, 2004, pp. 245–257.

[17] I. Corp., "Ibm tivoli service request manager." [Online]. Available: www-03.ibm.com/software/products/en/servicerequestmanager/

[18] B. Corp., "Bmc remedy9 service suite." [Online]. Available: www.bmc.com/it-solutions/remedy-itsm.html

[19] J. Zhang, L. Renganarayana, X. Zhang, N. Ge, V. Bala, T. Xu, and Y. Zhou, "Encore: Exploiting system environment and correlation information for misconfiguration detection," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 687–700, 2014.

[20] L. Pasquale, J. Laredo, H. Ludwig, K. Bhattacharya, and B. Wassermann, "Distributed cross-domain configuration management," *Service-Oriented Computing*, pp. 622–636, 2009.

[21] J. M. Such and N. Criado, "Resolving multi-party privacy conflicts in social media," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1851–1863, 2016.

[22] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*. IEEE, 2008, pp. 1–7.

[23] J. Ahrenholz, "Comparison of core network emulation platforms," in *Military Communications Conference, 2010-MILCOM 2010*. IEEE, 2010, pp. 166–171.