MeshScope: a bottom-up approach for ***configuration inspection*** in service mesh

Xing Li[1, 2], Xiao Wang[2], and Yan Chen[2]

[1]Zhejiang University, [2]Northwestern University

Introduction

As an emerging deployment paradigm for cloud-native applications, service mesh has developed rapidly in recent years and gained widespread attention in academia and industry. However, the administrator's management intention and configuration may not be consistent with actual system behavior. Unfortunately, current configuration validation and audit works only focus on the correctness of system configuration, but can not detect and deal with the inconsistency between the actual system behavior and the service mesh configuration.

In this paper, we present MeshScope, a bottom-up approach that can inspect the configuration in service mesh from the perspective of system behavior, and report the actual system behavior to the administrator to guide the subsequent configuration. It includes a novel hybrid policy verification mechanism which combines static and dynamic methods to examine the configurations from the data plane of service mesh, and a system behavior analysis which parses all the inconsistencies found and describes the system behavior in terms of traffic management and security.

What is service mesh?

A dedicated infrastructure layer that manages the communication among microservices.

- **Data Plane**: A series of sidecars responsible for proxying and managing inbound and outbound traffic for services.
- **Control Plane**: A centralized program that receives management policies and controls the behavior of the data plane accordingly.

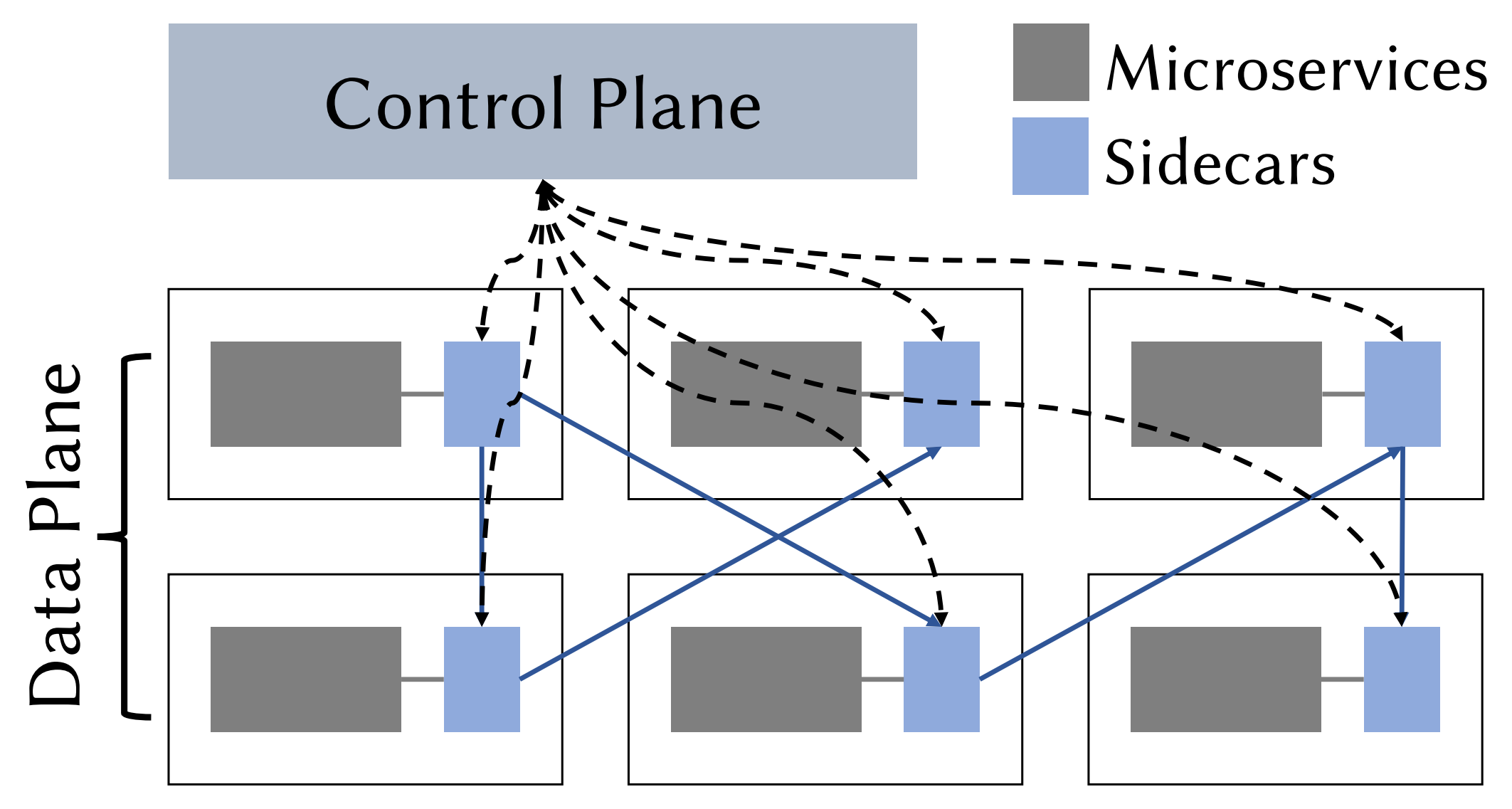


Figure 1. Service Mesh Architecture

Why is configuration inspection necessary?

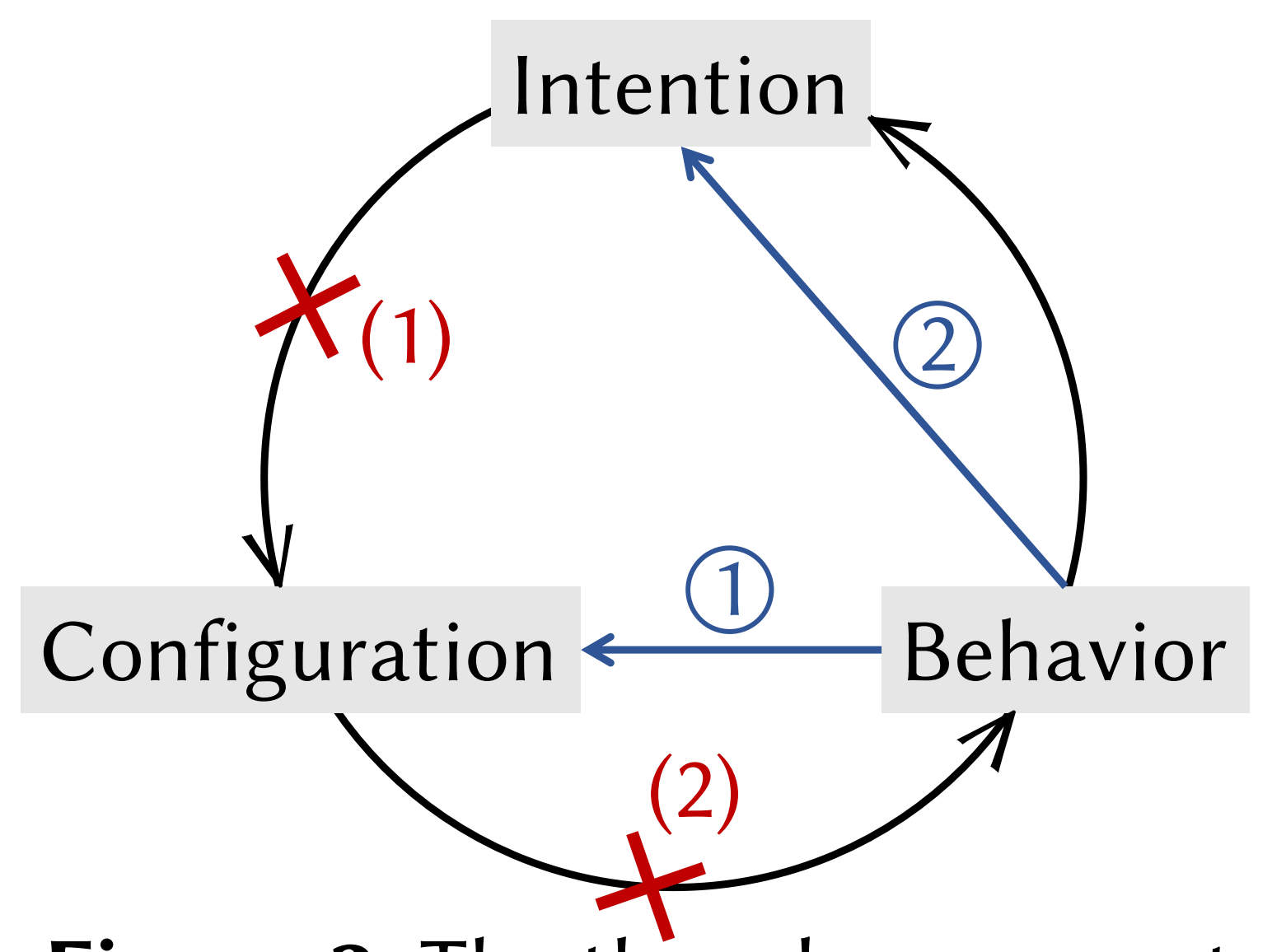


Figure 2. The three key concepts in service management.

It is difficult to achieve consistency between intention, configuration, and behavior.

(1) Administrators' management intentions may not be configured correctly.

a) The complexity of service management.

b) The un-intuitiveness of policy configuration.

(2) The configured policies may not be reflected in the behavior of data plane.

a) Inconsistency between control and data plane.

b) Installed policies may not be enforced.

Current solutions (configuration validation & audit) only aim at issue (1).

What is MeshScope?

① Inspecting the configuration with system behavior.
② Providing a mesh behavior view guide configuration.

- **A hybrid policy verification mechanism.**
  - A static verification to detect the inconsistencies between control plane and data plane.
  - A dynamic verification to check policy enforcement.
- **A system behavior analysis mechanism.**
  - Analyzing the collected inconsistencies.
  - Describing the system behavior.

Q: Why is it challenging in service mesh? (Compared with active probe testing work in SDN)

A: The complexity of service mesh policies introduces new challenges to both the generation of test workloads and the verification of system behavior.

- Various types of policies and the combination of rich matching conditions (e.g., API, method, version, etc.) make it difficult to generate high-quality detection requests.
- Various policy actions, such as load balancing, authorization, and weighted routing, can lead to challenges in deducing and verifying system behavior.

⭐ *The sidecar's computing power is much stronger than switch so that we can achieve efficient distributed testing and meet the performance requirements in practice.*

How does MeshScope work?

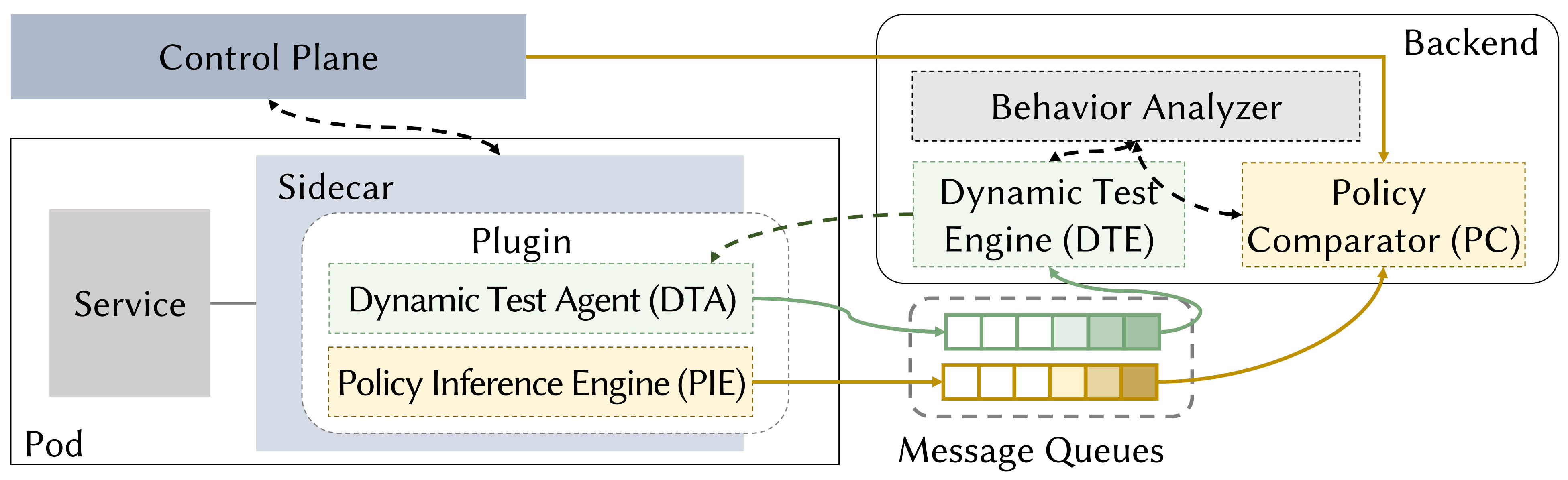


Figure 3. MeshScope Architecture

Architecture
- A **plugin** embedded in each sidecar for performing tests
- A **backend** for managing the tests and analyzing the system behavior
- A series of **message queues** for caching test results.

Policy Verification

I. Continuous static policy verification:

i. The PIE extracts the configuration from the sidecar, deduces the corresponding control plane policies, and sends them to the PC.

ii. The PC compares the policies from different planes to identify inconsistencies.

II. On-demand dynamic policy verification:

i. The DTE issues dynamic verification instructions.

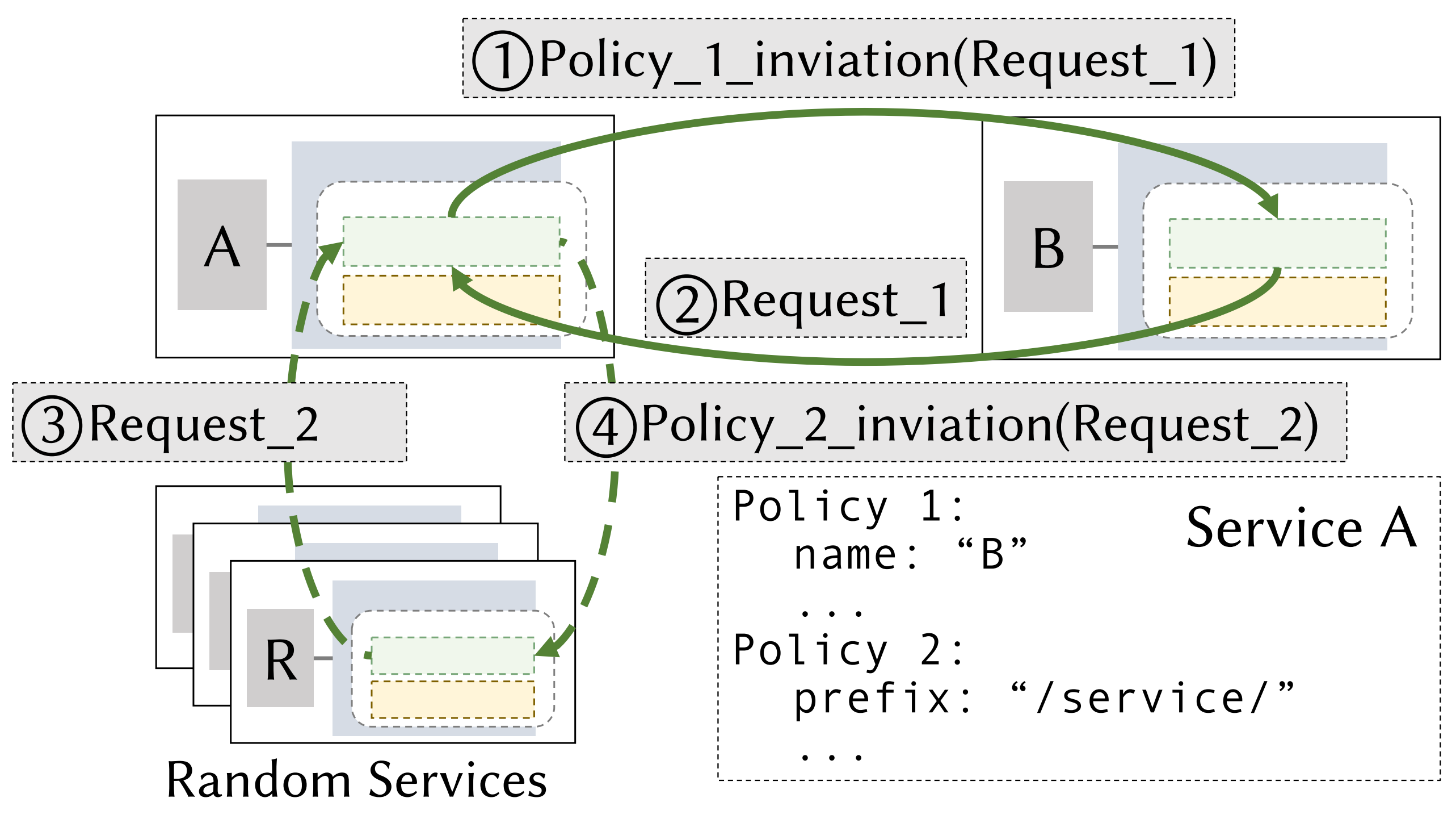
ii. The DTAs conduct distributed tests and send identified anomalies to the backend.

Behavior Analysis

I. Identifying root causes of the collected anomalies.

II. Providing preliminary repair suggestions based on some insights.

III. Describing system behavior in terms of traffic management and security.


①Policy_1_inviation(Request_1)
A
B
②Request_1
③Request_2
④Policy_2_inviation(Request_2)
R
Random Services
Service A

```
Policy 1:
  name: "B"
  ...
Policy 2:
  prefix: "/service/"
  ...
```

Figure 4. An example of dynamic verification for inbound policies: to verify Policy 1 for service A, its *Dynamic Test Agent* sends an invitation to the expected sender (B) (①). Afterwards, the DTA of B sends back the request A needs to check whether Policy 1 has been enforced (②). For the policies with no specific desired sender, such as Policy 2, the receiver sends invitations to random services (③, ④). Finally, agents send the failures to the backend for analysis.

Ongoing and future work

Currently, we are working on the implementation of the proposed policy verification mechanism. In future work, we aim to utilize emerging technologies to investigate the identified inconsistencies, and automate the diagnosis and repair of misconfigurations.