The 20th International Conference on Software Engineering Advances
September 28, 2025 to October 02, 2025 - Lisbon, Portugal

Protocol-aware Cloud Gateway with Adaptive Rate Control

Ivana Kovačević *, Vasilije Milić, Isidora Knežević, Tamara Ranković, Miloš Simić

Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia

* **kovacevic.ivana@uns.ac.rs**

About the presenter

Current: PhD student and TA at the Faculty of Technical Sciences, University of Novi Sad, and a researcher on the TaRDIS project

Education: Master of Science from the Faculty of Technical Sciences and a Bachelor with Honours in Computer and Control Engineering from the Faculty of Technical Sciences, University of Novi Sad, Serbia

Research interests: multi-party computation, distributed systems, cybersecurity

About the project

- **T**rustworthy **a**nd **R**esilient **D**ecentralised **I**ntelligence for Edge **S**ystems

- TaRDIS focuses on supporting the correct and efficient development of applications for swarms and decentralised distributed systems, by combining a novel programming paradigm with a toolbox for supporting the development and execution of applications.

- TaRDIS's primary goal is to significantly ease the complexity and reduce the effort of building correct and efficient heterogeneous swarms.

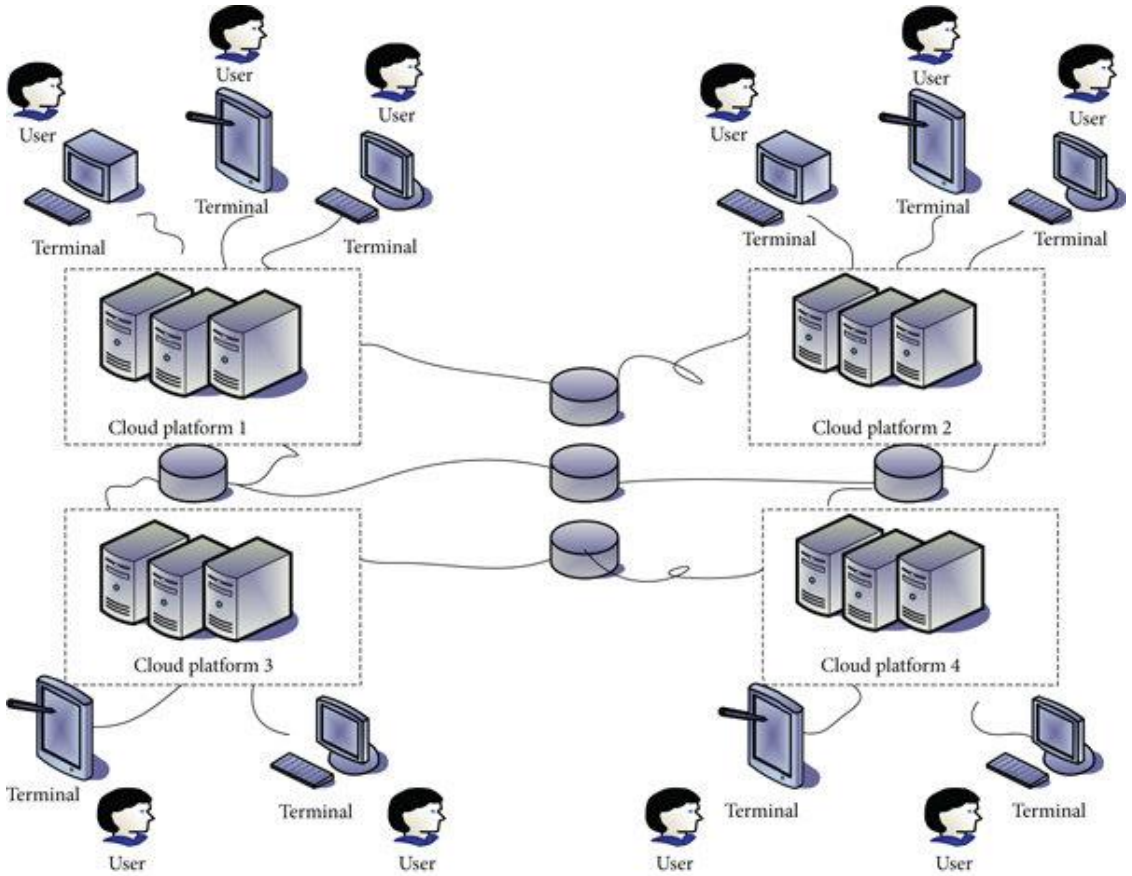
TaRDIS

Agenda

- Introduction
- Motivation
- Proposed prototype
- Architecture
- Protocol Transcoding (HTTP ↔ gRPC)
- Rate limiter
- Request flow
- Evaluation
- Conclusion

Introduction

- Distributed cloud systems must support a wide range of data formats and communication protocols.
- Current cloud solutions are primarily optimized for inter-service communication via RPC (binary), which may not always be suitable for external web clients.
- Communication protocols play a pivotal role in:
  - Real-time data conversion
  - Ensuring interoperability without data loss
  - Meeting latency and reliability constraints.



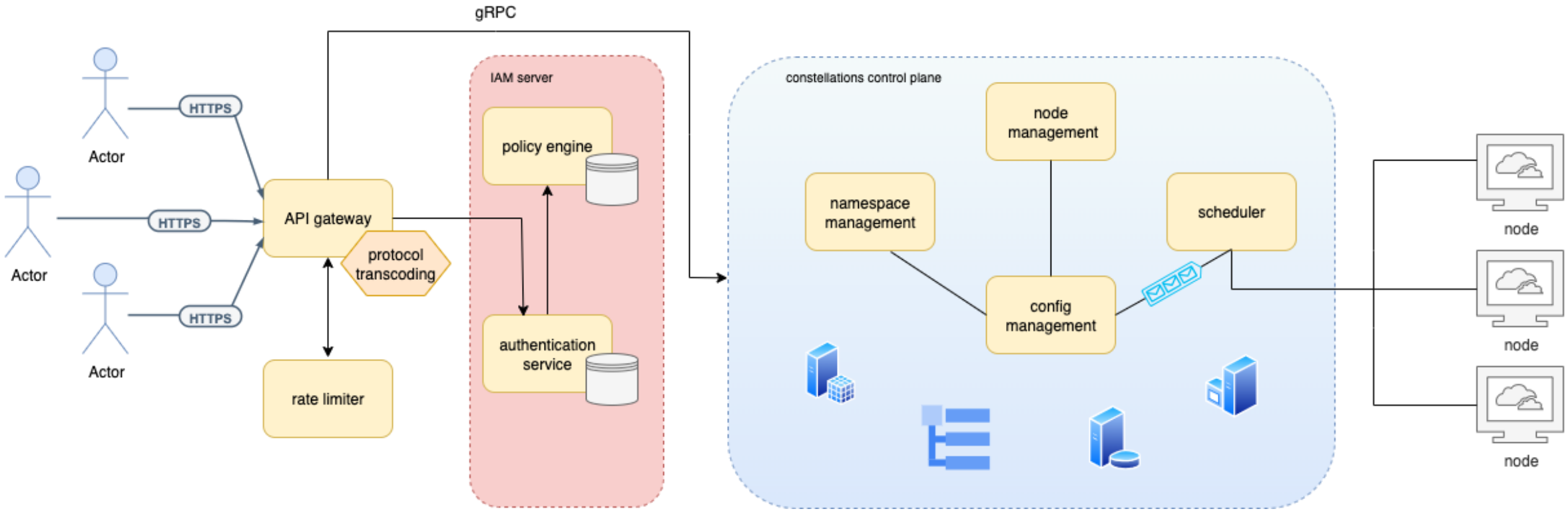
Motivation

- Cloud services require high availability, low latency, and robustness, as well as seamless integration across applications, users, and services.
- Research questions:
  - How to achieve robustness while enabling support for multiple protocols in the same environment?
  - How to ensure high availability and low latency, without having network congestion and resource exhaustion due to high traffic and a high frequency of requests?
- Goals:
  - ✓ Seamless protocol transcoding (HTTP ↔ gRPC)
  - ✓ Centralized and configurable rate-limiting
  - ✓ Preserve system scalability and availability.

Proposed prototype

- Open-source, platform-independent components developed in Golang for distributed cloud environments, integrated into Constellations (c12s) distributed cloud platform.
- **Protocol-aware Gateway**
  - Transcodes HTTP → gRPC dynamically using client discovery and reflection
  - RESTful compliance without modifying service code
  - Ensures no data loss, as well as proper headers and response code conversion
- **Rate Limiting Service**
  - Supports the Token bucket, the Leaky bucket, and the Sliding window algorithms
  - System-wide, user-level, and priority-based rate control

Architecture



Protocol Transcoding (HTTP ↔ gRPC)

- REST APIs dominate (80% of public APIs) – there is a need for gRPC to follow REST patterns.
- The process of HTTP route generation consists of:
  1. Generation of sub-routers for every group,
  2. Sub-routing groups based on the version,
  3. Assigning a path to each route based on the method name from the configuration file,
  4. Creating a middleware that integrates a handler function and HTTP method type for each route.

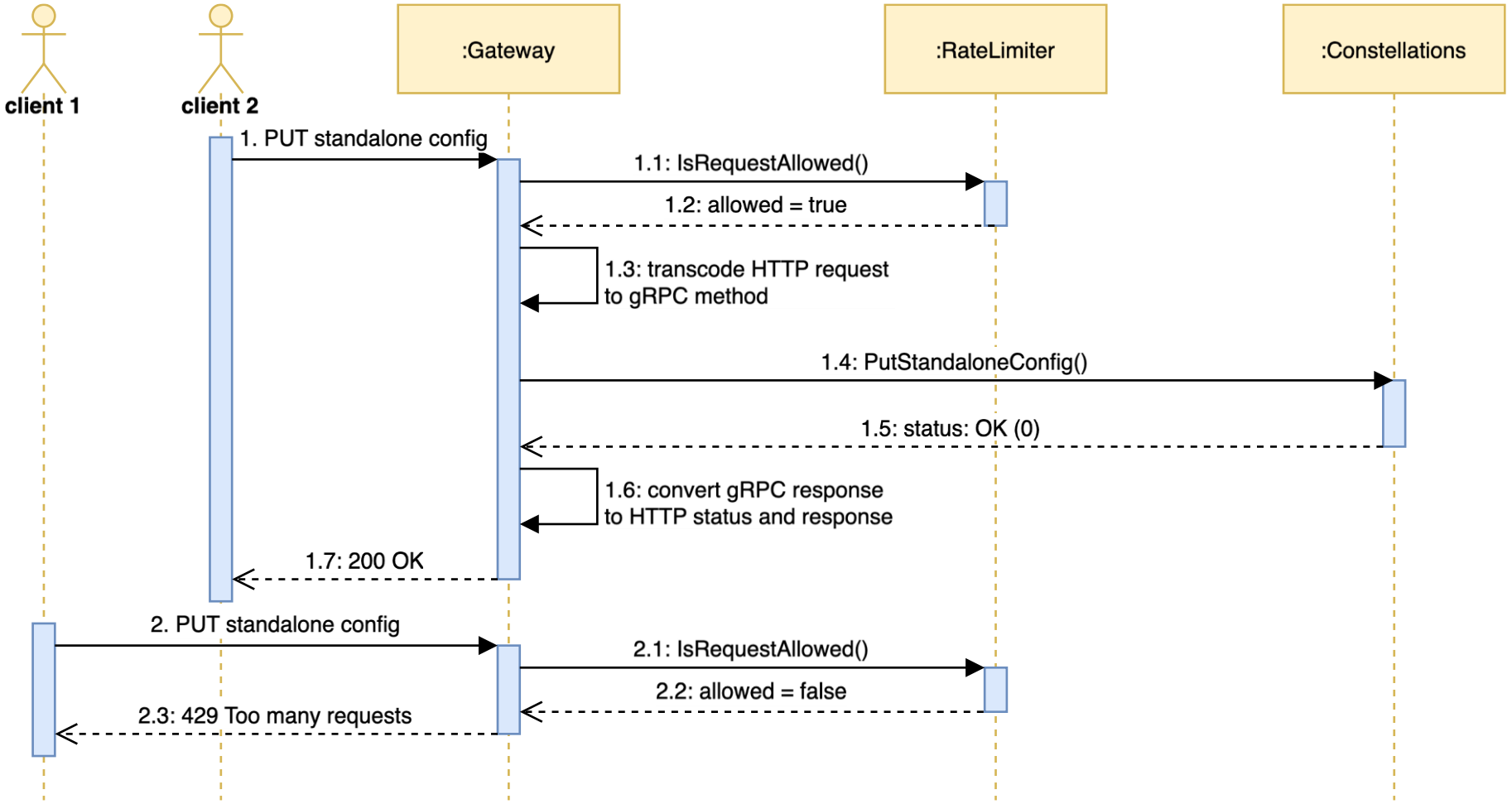
```yaml
gateway:
  route: /apis
  port: 5555
services:
  Kuiper: kuiper:5000
  ExampleService: example:9001
  RateLimitService: rate_limiter_service:8080
groups:
  core:
    v1:
      CreateExample:
        method_route: /example-route
        type: POST
        service: ExampleService
      PutStandaloneConfig:
        method_route: /configs/standalone
        type: PUT
        service: Kuiper
```

Example of a YAML gateway configuration

Rate limiter

- Per-client customization - flexible limits for end users and service methods; if the user is omitted, the rate limiter applies globally at the system level.
- Algorithms supported – Token Bucket (traffic shaping), Leaky Bucket (smooth throughput), Sliding Window (fine-grained limits).
- The rate limiter service offers configurable attributes:
  - Unique limiter ID – format user_id-method_id (regex supported)
  - REQ_LIMIT + PERIOD → total allowed requests
  - BURST → max concurrent requests (throttling)
  - PRIORITY → queueing (lower value = higher priority)
  - Caching integration for scalability

Request flow



Evaluation

- Test setup: 1000 requests to the same route, configured to use the token bucket, leaky bucket, and sliding window algorithm, with the same REQ_LIMIT parameter set to 10.
- Average latency - the ratio between regular response time and response time when the rate limiter is applied.
- Control rate - the ratio between the number of successful and the number of rejected requests.
- The transcoding process occurs at the beginning of the request call, and it takes less than 5ms.

	Token Bucket	**Leaky Bucket**	**Sliding Window**
Avg. response time	0.097s	1.001s	0.095s
Avg. latency	0.074s	0.043s	0.022s
Control rate	0.1273	/	0.1235
Total time (~1000 req)	10.502s	13.253s	10.084s

Conclusion

- We proposed platform-agnostic, easy-to-integrate components:
  - protocol-transcoding gateway and
  - adaptive rate-control service.
- We achieved high availability and robustness through consistent communication, ensuring no performance loss, and fair resource allocation for users and services.

Future work:

- Adaptive rate limiting using telemetry & monitoring
- Support for distributed rate-limiting
- Broader validation through real-world integration

Thank You!

Questions?