Sentry: QoE-Aware Failure Handling in Large-Scale Overlay Networks via Spatiotemporal GNNs

Xingxing Yang
Tsinghua University
Beijing, China
yangxx22@mails.tsinghua.edu.cn

Bo Wang
Tsinghua University
Beijing, China
wangbo2019@tsinghua.edu.cn

Wufan Wang
Beijing University of Posts and Telecommunications
Beijing, China
wufanwang@bupt.edu.cn

Yixin Shen
Tsinghua University
Beijing, China
syx@ieee.org

Minhu Wang
Tsinghua University
Beijing, China
minhuw@acm.org

Wangqiu You
ByteDance China
Shanghai, China
youwangqiu@bytedance.com

Linhui Lou
ByteDance China
Shanghai, China
loulinhui.shey@bytedance.com

Pei Xu
ByteDance China
Shanghai, China
xupei.os@bytedance.com

Lihang Gao
ByteDance China
Shanghai, China
gaolihang.123@bytedance.com

Zongzhi Hou
ByteDance China
Shanghai, China
houzongzhi.2023@bytedance.com

Mingwei Xu
Tsinghua University
Beijing, China
xumw@tsinghua.edu.cn

Abstract

With the widespread deployment of large-scale overlay networks, transmission failures among edge servers occur frequently, severely impacting service availability and Quality of Experience (QoE). Existing failure handling solutions struggle to effectively address such scenarios. Accordingly, we propose Sentry, a QoE-aware failure handling method based on spatiotemporal graph neural networks. Instead of relying on precise localization of specific faults or root causes, Sentry directly identifies and offlines a set of edge servers related to failures to ensure service level objective (SLO) while minimizing QoE degradation. Sentry constructs a state graph based on real relay streams, combining temporal modeling and graph neural networks to effectively filter out short-term network fluctuations and transform the complex combinatorial optimization problem into an efficient node-level prediction task. The experimental results show that Sentry outperforms existing solutions in both success rate and cost of failure handling, demonstrating strong overall performance and deployment potential.

CCS Concepts

• **Networks → Network management**; **Network monitoring**; • **Computing methodologies → Machine learning**.

Keywords

Failure Handling; Quality of Experience; Graph Neural Networks; Network Monitoring; Large-Scale Overlay Networks

ACM Reference Format:
Xingxing Yang, Bo Wang, Wufan Wang, Yixin Shen, Minhu Wang, Wangqiu You, Linhui Lou, Pei Xu, Lihang Gao, Zongzhi Hou, and Mingwei Xu. 2025. Sentry: QoE-Aware Failure Handling in Large-Scale Overlay Networks via Spatiotemporal GNNs. In *Proceedings of the ACM CoNEXT-2025 Workshop on Computing and Network Convergence (CNC '25), December 01–04, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3769270.3770125

1 Introduction

With the rapid development of various online applications such as video conferencing [14], cloud gaming [20], and live streaming [8], users demand higher quality of network services. The best-effort transmission model used in traditional Internet architecture struggles to ensure stable service under complex traffic patterns and dynamic link conditions. Therefore, large-scale overlay networks have become essential infrastructure for supporting a wide range of daily applications [1, 2, 13]. These networks deploy edge data centers worldwide, each equipped with multiple edge servers, to optimize transmission performance and enhance service reliability and availability.

However, this highly distributed architecture also introduces complex network interconnection challenges among edge servers

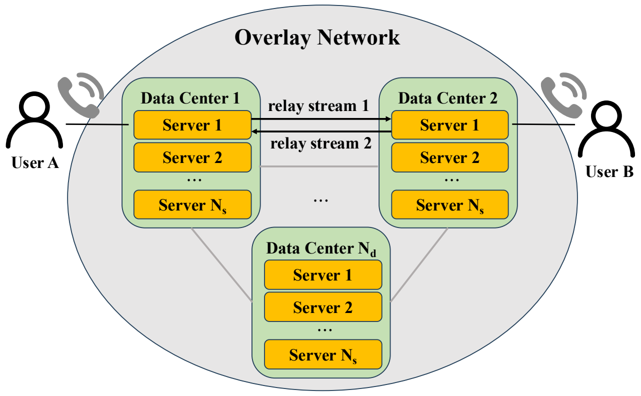


Figure 1: Relay streams in an audio call.

(detailed in Section 2.1). Transmission failures frequently occur, especially in environments with numerous servers deployed across multiple cloud providers and geographic regions, posing a serious threat to overall network availability. For example, in a real-time video call between two users, transmission failures between edge servers may cause audio-video desynchronization or even connection drops, significantly affecting user experience [9].

To ensure high availability of the overlay network, the system must continuously meet stringent SLO. For example, it is common practice to require that the success rate of relay streams not fall below 99.99%; otherwise, a transmission failure is considered to have occurred, and failure handling should be completed within a short time. Here, relay streams refer to data streams between edge servers that carry user communications. As shown in Figure 1, if users A and B are connected to Server 1 in Data Center 1 and Server 1 in Data Center 2 respectively, their audio communication generates two relay streams relying on the stable forwarding capabilities between these two edge servers. Once the underlying connection fails to stay alive, or the QoS metrics of the stream (such as latency or packet loss rate) become abnormal, the stream is classified as failed; otherwise, it is considered successful. The overall SLO compliance rate of the system (i.e., the proportion of time the SLO is met) directly reflects the availability of the overlay network. Therefore, automated, precise, and real-time failure handling mechanisms have become a core challenge in the development of overlay networks.

Unfortunately, existing failure handling methods cannot effectively address transmission failures between edge servers in large-scale overlay networks. Current mainstream methods can be roughly divided into two categories: one relies on accurately locating the failure before avoidance and repair [5, 12, 18, 22, 24, 25, 29, 30]; the other dynamically selects overlay paths based on real-time path observations to bypass potentially faulty areas [4, 13, 19, 21, 23]. The former faces challenges in failure localization and diagnosis, struggling to pinpoint specific failure locations and prone to false positives. The latter is limited by the computational complexity of path selection and the accuracy of performance measurements, making it difficult to promptly detect performance degradation of individual servers within a data center and to perform fine-grained scheduling (detailed in Section 2.2).

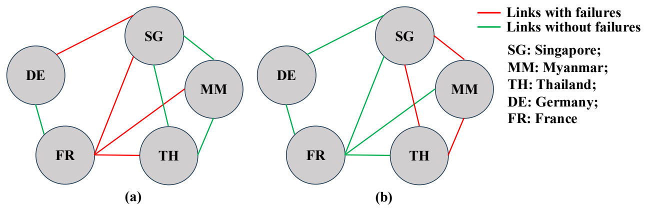


Figure 2: Overlay network transmission failure cases. (a) Backbone failure disconnects same-provider datacenters across regions; (b) Regional failure disconnects same-provider datacenters locally.

To address the above challenges, we propose Sentry, a QoE-aware failure handling solution based on spatiotemporal graph neural networks. Rather than relying on precise localization of specific failure points or root causes, Sentry directly identifies and offlines a set of edge servers related to transmission failures, thereby ensuring the overall stream success rate meets SLO requirements while minimizing unnecessary server offlining and maintaining high QoE. Sentry incorporates several key designs: First, it employs a Temporal Embedding Module to capture the temporal evolution of server states, enabling differentiation between short-term fluctuations and persistent failures, thereby improving identification accuracy. Second, it constructs a sparse graph based on relay streams among servers and applies graph neural networks to score each server node. This transforms the original combinatorial optimization problem into a node-level prediction problem, drastically reducing the search complexity from combinatorial to linear, significantly enhancing failure handling efficiency and generalization capability. Furthermore, Sentry decouples failure handling from path optimization. It passively handles failures based on the performance of real relay streams, then an independent overlay routing module uses active probing to perform path selection, balancing observation accuracy with system overhead.

We implemented Sentry and evaluated its performance in a simulated environment. The results demonstrate that Sentry achieves the highest failure handling success rate, thereby improving the overall SLO compliance rate. Although the number of offline servers is slightly higher than that of heuristic baseline methods, the difference is minimal, and the user experience loss caused by offlining is lower, showing superior overall performance and practical value. In the future, Sentry will be deployed in the large-scale overlay network of a globally leading network transmission service provider to further validate its effectiveness in real-world environments.

This work does not raise any ethical issues.

2 Background

This section first introduces common types of transmission failures in large-scale overlay networks, and then analyzes the limitations of existing failure handling methods.

2.1 Types of Transmission Failures

In large-scale overlay networks, transmission failures between edge servers exhibit diverse causes and complex manifestations. Failures may stem from hardware malfunctions of individual edge servers,

Table 1: Failure statistics of our overlay network

Failure Type	Failure Num.	Percentage
Server-level failures	74	31.5%
Data center-level failures	161	68.5%

or from broader issues such as international link congestion, submarine cable cuts, or regional network incidents—leading to simultaneous degradation of transmission quality between multiple edge data centers (as shown in Figure 2). In practice, different types of failures often occur concurrently and compound each other, significantly increasing the complexity of failure handling and severely impacting network availability and user experience.

2.2 Limitations of Existing Solutions

Failure handling solutions mainly fall into two categories:

The first category of solutions relies on accurate failure localization [5, 12, 18, 22, 24, 25, 29, 30], followed by path rerouting and failure reporting based on the identified faulty components. The effectiveness of such methods heavily depends on the accuracy of failure localization, which is particularly challenging in large-scale overlay networks. First, **failure localization itself is inherently difficult**. In large-scale overlays, the vast number of edge servers, the involvement of multiple infrastructure providers, and the presence of cross-border links make it hard to map performance anomalies to specific underlying components. Second, **failure determination is highly uncertain**. Transient network fluctuations and persistent failures often exhibit similar symptoms, making it difficult for the system to respond promptly without sacrificing accuracy. An overly aggressive policy may result in false positives and unnecessary rerouting, while a conservative one may delay failure mitigation and reduce system availability.

The second category bypasses failures via overlay routing [4, 13, 19, 21, 23] without explicit failure localization, dynamically selecting optimal paths based on real-time performance to avoid faulty regions. However, these methods face significant challenges in large-scale overlay networks. First, **the computational cost of path selection increases significantly**. In large-scale overlay networks, the number of edge servers is massive—reaching tens of thousands—and the algorithm for optimal path selection has high complexity, resulting in substantial scheduling overhead. Therefore, performing path planning at the granularity of individual edge servers is difficult to implement in real-world systems. To mitigate this, existing solutions often perform path planning at the edge datacenter level. While this reduces computational overhead to some extent, it fails to capture transmission anomalies associated with individual servers within a datacenter. In practice, when certain servers experience degraded performance, inter-datacenter measurements may not show noticeable deterioration, making it difficult for the system to detect and bypass the affected servers. Conversely, in some cases, the entire datacenter may be bypassed due to a few abnormal servers, leading to underutilization of healthy servers and reduced overall resource efficiency. As shown in Table 1, from January to March 2025, 31.5% of online transmission failures in our overlay network were related to transmission anomalies associated with individual servers within edge datacenters, highlighting the prevalence and severity of this issue. Second, **performance measurement of paths is inherently limited**. Active probing (e.g., ICMP Ping [17]) results may deviate from the actual performance experienced by real application traffic. Passive observation relies on existing traffic and is limited by traffic distribution, making it difficult to cover all paths.

3 Method Design

This section presents our method, Sentry. To capture the spatial relationships between servers and their temporal dynamics, Sentry adopts a spatiotemporal graph neural network (GNN) architecture. It integrates Gated recurrent unit (GRU) [7] for temporal modeling, GraphSAGE [11] for spatial aggregation, and utilizes a multi-layer perceptron (MLP) to generate an offlining score for each server, based on which the set of servers to be offlined is determined. We next describe the task definition, graph construction, model architecture, and the training and deployment details.

3.1 Problem Overview

To address transmission failures in large-scale overlay networks, we propose a server offlining-based failure handling approach, and formulate it as a graph-based node-level decision problem. In each decision round, the system predicts an offlining score for every edge server with failed streams based on the current state of relay streams and server-level features. A subset of servers is then selected for offlining, aiming to meet SLO as quickly as possible while minimizing the number of offline servers and preserving high QoE.

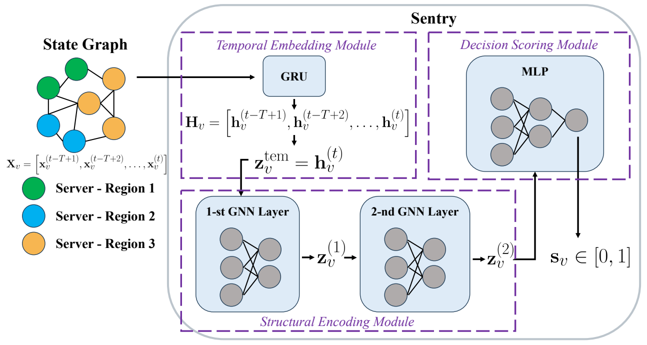
3.2 Graph Construction

To capture the structural correlations between edge servers in relay streams, we construct a graph $G = (V, E)$, where each node represents an edge server and each edge corresponds to a pair of servers that are connected by relay streams within the current time window. Specifically, the node set V includes all edge servers currently involved in relay streams. An undirected edge $(u, v) \in E$ is added if there exist relay streams between servers u and v. The resulting graph is sparse and reflects the connectivity topology formed by relay streams in the overlay network at the current time. For each edge $(u, v) \in E$, let $S_{uv} \geq 0$ denote the number of successful relay streams and $F_{uv} \geq 0$ the number of failed relay streams. The overall success ratio S is defined as:

$$S = \frac{\sum_{(u,v)\in E} S_{uv}}{\sum_{(u,v)\in E} (S_{uv} + F_{uv})} \tag{1}$$

Each node $v \in V$ is associated with a sequence of historical feature vectors $\{x_v^{(t-T+1)}, \ldots, x_v^{(t)}\}$, where each vector $x_v^{(t')} \in \mathbb{R}^5$ includes the following key features:

- fail_ratio: the proportion of failed relay streams among all relay streams associated with the server;
- mean_rtt: the average round-trip time (RTT) across all links of the server;



Sentry
State Graph
Temporal Embedding Module
GRU
$\mathbf{H}_v = [\mathbf{h}_v^{(t-T+1)}, \mathbf{h}_v^{(t-T+2)}, \ldots, \mathbf{h}_v^{(t)}]$
$\mathbf{z}_v^{\text{tem}} = \mathbf{h}_v^{(t)}$
$\mathbf{x}_v = [\mathbf{x}_v^{(t-T+1)}, \mathbf{x}_v^{(t-T+2)}, \ldots, \mathbf{x}_v^{(t)}]$
Decision Scoring Module
MLP
Server - Region 1
Server - Region 2
Server - Region 3
1-st GNN Layer
2-nd GNN Layer
$\mathbf{z}_v^{(1)}$
$\mathbf{z}_v^{(2)}$
$\mathbf{s}_v \in [0,1]$
Structural Encoding Module

Figure 3: Overview of the Sentry model architecture.

- mean_loss: the average packet loss rate across all links of the server;
- active_user_count: the current number of active users on the server;
- coverage_impact = $\frac{1}{N_v}$: the impact of offlining this server on regional coverage, where N_v denotes the total number of edge servers in the same region.

The feature set captures both the severity of server failures and the potential cost of offlining, covering key dimensions for offlining decisions. All features can be collected in real time by existing systems, ensuring good deployability.

3.3 Model Architecture

To capture temporal dynamics and structural dependencies, we design a spatiotemporal GNN combining GRU for temporal modeling and GraphSAGE for spatial aggregation, driven by two key motivations: (1) **temporal modeling** aims to distinguish persistent failures from short-term fluctuations, avoiding misjudgments and unnecessary server offlining caused by transient network anomalies such as sudden traffic bursts leading to temporary congestion. To achieve this, a Temporal Embedding module is introduced to model the state sequence of each server over the past T time steps, capturing the evolution trend of states and enhancing failure discrimination robustness; (2) **structural awareness** helps capture correlated failures and improve decision quality. In real networks, transmission failures often affect multiple servers due to shared root causes. Relying only on local information may mistake such anomalies for independent issues, leading to excessive offlining. For example, if a link issue causes high packet loss between servers A and B, offlining either one may suffice—without structural awareness, both might be offlined unnecessarily, wasting resources and reducing coverage. By aggregating neighbor states through a GraphSAGE-based Structural Encoding module, the model detects structural anomalies and avoids redundant actions.

As shown in Figure 3, the model consists of three main modules. First, the **Temporal Embedding Module** takes as input the state feature sequence of each node v in the state graph G over the past T time windows:

$$\mathbf{X}_v = \left[\mathbf{x}_v^{(t-T+1)}, \mathbf{x}_v^{(t-T+2)}, \ldots, \mathbf{x}_v^{(t)}\right], \quad \mathbf{x}_v^{(t')} \in \mathbb{R}^5.$$

We employ a single-layer GRU to capture the temporal dynamics of each node, modeling how its state evolves over time—whether it is persistently deteriorating, gradually recovering, or merely undergoing short-term fluctuations. The GRU produces a sequence of hidden states

$$\mathbf{H}_v = \text{GRU}(\mathbf{X}_v) = \left[\mathbf{h}_v^{(t-T+1)}, \mathbf{h}_v^{(t-T+2)}, \ldots, \mathbf{h}_v^{(t)}\right],$$

and the last hidden state is used as the temporal-aware representation of node v: $\mathbf{z}_v^{\text{tem}} = \mathbf{h}_v^{(t)}$.

Next, on the state graph G, the **Structural Encoding Module** uses each node temporal-aware representation $\mathbf{z}_v^{\text{tem}}$ as input and applies two layers of GraphSAGE to perform structure-aware representation learning. GraphSAGE uses mean aggregation over neighbors for efficient local modeling in sparse graphs, followed by ReLU activation to enhance nonlinearity. The update rule at the lth layer is given by

$$\mathbf{z}_v^{(l)} = \phi\left(\mathbf{W}^{(l)} \cdot \text{MEAN}\left(\left\{\mathbf{z}_v^{(l-1)}\right\} \cup \left\{\mathbf{z}_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right)\right),$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node v, $\phi(\cdot)$ is the ReLU activation function, and $\mathbf{W}^{(l)}$ is the learnable weight matrix of the lth layer. Each node finally obtains a low-dimensional embedding that fuses both temporal and structural information: $\mathbf{z}_v^{\text{str}} = \mathbf{z}_v^{(2)}$.

The **Decision Scoring Module** takes the embedding $\mathbf{z}_v^{\text{str}}$ of each node with failed relay streams and computes an offlining score via a two-layer MLP with ReLU and Sigmoid activations, indicating the urgency of offlining node v:

$$s_v = \sigma\left(\mathbf{W}_2 \cdot \phi(\mathbf{W}_1 \cdot \mathbf{z}_v^{\text{str}} + \mathbf{b}_1) + \mathbf{b}_2\right), \quad s_v \in [0,1],$$

where $\mathbf{W}_1, \mathbf{W}_2$ and $\mathbf{b}_1, \mathbf{b}_2$ are learnable parameters, $\phi(\cdot)$ denotes ReLU, and $\sigma(\cdot)$ denotes Sigmoid.

3.4 Training and Deployment

To achieve efficient and deployable offlining decisions, we use offline training with online inference.

3.4.1 Supervision and Labels. We treat offlining as supervised learning on production failure logs, using state feature sequences of servers with relay streams as inputs. A server is labeled positive (label 1) if offlining the server improves QoE under the SLO without a lower-cost alternative achieving the same; otherwise, it is labeled negative (label 0). The model is trained with binary cross-entropy loss to minimize the gap between predicted scores s_v and labels y_v.

3.4.2 Deployment and Inference. During deployment, the model is periodically updated offline. At inference, it takes the recent state features of each server and the current graph as input, and outputs an offlining score $s_v \in [0,1]$ for each server that has failed streams. Servers are ranked by score in descending order and added one by one to the offlining set. After each addition, the server is removed and the stream success rate checked. The process stops when the SLO is met, and the resulting set forms the offlining plan. The model includes a GRU layer (hidden dimension 32), two GraphSAGE layers (hidden dimension 64), and an MLP layer (hidden dimension 32). The output is a Sigmoid-activated scalar. The model is trained with Adam optimizer (learning rate 0.001) for 10 epochs. All simulations use this configuration.

4 Evaluation

We evaluate Sentry through simulations. Section 4.1 details the setup, and Section 4.2 compares Sentry with baselines.

4.1 Experiment Setup

This section describes the topology, trace generation, baselines, and evaluation metrics used in our simulations.

4.1.1 Traces. We implemented a Python-based trace generator to construct a server-level overlay network dataset. The simulated network includes 10 service regions and 30 edge data centers (2 servers each). Five major regions have 5 data centers each, while the others have 1. Each region operates as an independent service domain. In the failure-free scenario, we randomly sample 50% of all possible server pairs and generate end-to-end relay paths for each selected pair. The following network quality parameters are assigned: RTT is uniformly sampled from [1, 200] ms, packet loss rate from [0, 0.001], and the initial stream success rate is set to 100%. The number of streams between each server pair is uniformly sampled from the range [0, 10]. To support temporal modeling, we generate server state features for 3 consecutive observation windows (each lasting 30 seconds) for every failure scenario. Each server at each time step is described by a five-dimensional feature vector: fail_ratio, mean_rtt, mean_loss, active_user_count, and coverage_impact, where active_user_count is uniformly sampled from [1, 100]. All path quality parameters are re-sampled at each time step to reflect the dynamic dynamic changes in server state.

We design three typical failure scenarios based on the empirical relationship between packet loss rate and stream success rate fitted from production data:

$$y = \begin{cases} 1 & \text{if } 0 \leq x \leq 0.132 \\ -1.06x + 1.14 & \text{if } 0.132 < x \leq 0.5 \\ -1.2x + 1.2 & \text{if } 0.5 < x \leq 1 \end{cases} \tag{2}$$

where y denotes the success rate and x is the loss rate.

We design the following failure scenarios: (1) **Mild Path Failure Scenario**: 100 cases with failure ratios randomly selected between 0.1% and 1%. For each case, a corresponding number of edge server paths are randomly sampled, and their packet loss rates are set to random values within $(0.132, 1]$. Half of these cases are used for training, where beam search [16] generates multiple offlining solutions meeting the SLO; the one with the lowest failure handling cost is selected as a weak label. The other half form the test set to evaluate Sentry and baselines on failure handling success rate and cost; (2) **Moderate Path Failure Scenario**: 100 cases with failure ratios randomly selected between 1% and 10%, with randomly sampled edge server paths assigned packet loss rates in $(0.132, 1]$. Half are used as training data, and half as test data; (3) **Server-Wide Path Failure Scenario**: 100 cases with failure ratios randomly selected between 0.1% and 10%. For each case, a corresponding number of edge servers are randomly sampled, and all their associated paths have packet loss rates uniformly set to random values within $(0.132, 1]$. Half of the cases are used for training, and half for testing. All failure cases retained in the dataset are designed to trigger the failure handling logic of the system.

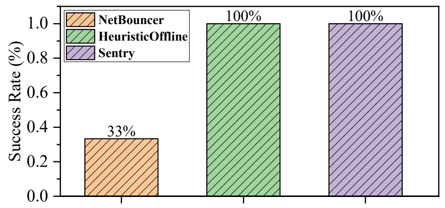


Figure 4: Comparison of failure handling success rates.

4.1.2 Baselines. (1) **NetBouncer [22]** is a failure detection mechanism originally designed for data centers. To adapt it to the overlay network scenario, the detection rule is modified such that when an edge server loses all paths with success rate 1, it is considered faulty and triggers offlining; (2) **HeuristicOffline** is a heuristic offlining strategy based on failure rates, where servers are sorted by fail_ratio from high to low and offlined sequentially until overall service recovers to meet the SLO.

For failure handling methods based on overlay routing, we attempted to build a full path rerouting mechanism at the server granularity, i.e., calculating the K-shortest loopless paths between any two edge servers using Yen's algorithm [27]. However, this method has a computational complexity of $O(n^5)$ for n nodes, failing to meet scalability requirements. Therefore, such methods are not included in the simulation. We plan to compare Sentry with overlay routing methods based on datacenter granularity in real system deployments to quantify performance advantage under fine-grained scheduling.

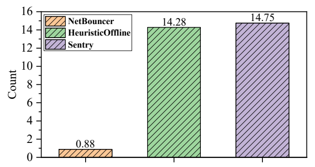
4.1.3 Metrics. We classify the evaluation metrics into two categories: (1) **Failure Handling Success Rate**, defined as the proportion of cases where the overall relay stream success rate (as defined in Equation 1) reaches 99.99% after applying a failure handling solution; and (2) **Failure Handling Cost**, which includes two components—namely, the number of offlined servers, reflecting potential redundancy in server shutdowns, and the QoE degradation score, quantifying the user experience impact. For each offlined server v, the local QoE loss is calculated as: $\text{QoE}_v = \alpha \cdot \text{active\_user\_count}_v + \beta \cdot \text{coverage\_impact}_v$, where $\alpha = 1.0$ and $\beta = 100$ represent the weights for user count and coverage impact, respectively. The overall QoE degradation score is computed by summing the local QoE losses across all offlined servers:

$$\text{QoE Degradation Score} = \sum_{v \in \text{offline\_set}} \text{QoE}_v.$$

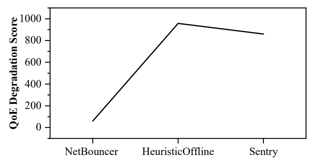
4.2 Comparison with Baselines

We systematically compare Sentry with baselines in simulation. The evaluation metrics include failure handling success rate, the number of offlined servers, and QoE degradation caused by server offlining.

First, as shown in Figure 4, Sentry achieves a 100% failure handling success rate across all 150 test cases, demonstrating excellent failure handling capability. In contrast, HeuristicOffline also achieves a 100% success rate, validating the effectiveness of prioritizing high-fail-ratio servers for offlining. However, NetBouncer achieves only 33.33%, significantly lower than the others, indicating



(a) Average server offlining counts per failure case.



(b) Average QoE loss caused by server offlining.

Figure 5: Comparison of failure handling costs.

that its datacenter-oriented design does not transfer well to the overlay network context.

In terms of offlining cost, as shown in Figure 5a, Sentry requires an average of 14.75 offlined servers, slightly higher than HeuristicOffline (14.28), but the difference is small. This indicates that Sentry maintains high success rates while avoiding excessive offlining. While NetBouncer shows the lowest offlining count (0.88), this comes at the cost of a significantly lower success rate, making it impractical for high-availability deployments.

Furthermore, as shown in Figure 5b, Sentry outperforms in QoE degradation score, with an average of 860.09—significantly lower than HeuristicOffline (957.88). This demonstrates that Sentry not only handles failures effectively but also intelligently selects servers with minimal user impact for offlining, thereby minimizing user experience degradation. While NetBouncer shows the lowest QoE degradation (59.89), this comes at the cost of failing to address failures, rendering the metric less meaningful.

Overall, Sentry achieves a strong balance between failure handling success rate and cost, demonstrating better overall performance and deployment potential. Compared to simple heuristics, Sentry leverages spatiotemporal GNN for smarter decisions; compared to traditional datacenter approaches, Sentry better adapts to the complexity of large-scale overlay networks.

To further validate Sentry in real-world scenarios, we plan to deploy and conduct online A/B testing in a large-scale overlay network operated by a leading global transmission service provider. The target network spans over 200 countries and territories, encompassing hundreds of edge data centers and tens of thousands of servers. The experiment will compare failure handling performance with and without Sentry, and against datacenter-level overlay routing methods, to assess the advantages of Sentry.

5 Discussion

Sentry complements, not replaces, existing failure handling mechanisms, but deployment still faces challenges. First, the update frequency must be carefully balanced to avoid system instability or missing critical changes [10]. Second, in real-world scenarios, failure data is scarce and user feedback is often delayed, which can negatively impact model quality. As future work, we will explore the integration of weak-label learning [28] and lightweight online fine-tuning [3], guided by online feedback.

Furthermore, after Sentry offlines abnormal servers, a large number of users need to reconnect. Without proper control, this may lead to traffic concentration and new failures. To address this, the system incorporates both user connection rate control and load-aware edge server mechanisms. The former staggers server removal to mitigate the herd effect [26], while the latter dynamically estimates regional capacity based on reported resource usage to prevent overload. Although this strategy is generally stable, its reliance on limited observations and empirical thresholds can lead to local imbalances in the presence of traffic surges or complex cross-region scheduling. Existing methods [6, 15] also struggle to handle such scenarios effectively. We plan to use machine learning to improve load prediction and scheduling.

6 Conclusion

We present Sentry, a spatiotemporal graph neural network-based failure handling approach designed to enhance transmission quality in large-scale overlay networks. By modeling relay streams between edge servers as a state graph, Sentry combines temporal modeling and spatial aggregation to accurately predict anomalous servers and make offlining decisions. This leads to significantly improved network availability and reduced failure handling cost. Experimental results demonstrate that Sentry outperforms existing methods across various failure scenarios, exhibiting strong robustness and generalization capabilities.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2024YFB2906705.

References

[1] 2022. Agora's Software Defined Real-Time Network™ Delivers Real-Time Internet Advantages Over Content Delivery Network. https://hello.agora.io/rs/096-LBH-766/images/Agora_WP_SD-RTN-Delivers-RealTime-Internet-Advantages.pdf.

[2] 2025. AWS Global Accelerator. https://aws.amazon.com/global-accelerator (accessed July 2025).

[3] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. 2019. Online continual learning with maximal interfered retrieval. *Advances in neural information processing systems* 32 (2019). https://dl.acm.org/doi/abs/10.5555/3454287.3455350

[4] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. 2001. Resilient overlay networks. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*. 131–145. doi:10.1145/502034.502048

[5] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. 2018. Odin:{Microsoft's} scalable {Fault-Tolerant}{CDN} measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 501–517. https://dl.acm.org/doi/10.5555/3307441.3307484

[6] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. 2015. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 167–181. doi:10.1145/2829988.2787500

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014). doi:10.48550/arXiv.1412.3555

[8] Jason Clements, Teodros Gessesse, Darshan Sedani, and Jerry Klein. 2016. Live video broadcasting mobile application for social sharing. US Patent App. 14/821,519.

[9] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review* 41, 4 (2011), 362–373. doi:10.1145/2043164.2018478

[10] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37. doi:10.1145/2523813

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017). https://dl.acm.org/doi/10.5555/3294771.3294869

[12] Rupa Krishnan, Harsha V Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. 2009. Moving beyond end-to-end path information to optimize CDN performance. In *Proceedings of the 9th ACM*

SIGCOMM conference on Internet measurement. 190–201. doi:10.1145/1644893.1644917

[13] Geng Li, Shuihai Hu, and Kun Tan. 2024. Panorama: Optimizing Internet-scale {Users'} Routes from End to End. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 935–949. https://dl.acm.org/doi/10.5555/3691992.3692049

[14] Bill Marczak and John Scott-Railton. 2020. Move Fast and Roll Your Own Crypto: A Quick Look at the Confidentiality of Zoom Meetings. https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/

[15] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19. doi:10.1145/1842733.1842736

[16] Peng Si Ow and Thomas E Morton. 1988. Filtered beam search in scheduling. *The International Journal Of Production Research* 26, 1 (1988), 35–62. doi:10.1080/00207548808947840

[17] Jon Postel. 1981. Internet Control Message Protocol. RFC 792. https://tools.ietf.org/html/rfc792

[18] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. 2017. Passive realtime datacenter fault detection and localization. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 595–612. https://dl.acm.org/doi/10.5555/3154630.3154679

[19] Ramesh K Sitaraman, Mangesh Kasbekar, Woody Lichtenstein, and Manish Jain. 2014. Overlay networks: An akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services* (2014), 305–328. doi:10.1002/9781118909690.ch16

[20] Ivan Slivar, Mirko Suznjevic, and Lea Skorin-Kapov. 2018. Game categorization for deriving QoE-driven video encoding configuration strategies for cloud gaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14, 3s (2018), 1–24. doi:10.1145/3132041

[21] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H Katz. 2004. OverQoS: An Overlay Based Architecture for Enhancing Internet QoS.. In *NSDI*, Vol. 4. 71–84. https://dl.acm.org/doi/10.5555/1251175.1251181

[22] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. {NetBouncer}: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 599–614. https://dl.acm.org/doi/10.5555/3323234.3323283

[23] Shengwen Tian, Jianxin Liao, Tonghong Li, Jingyu Wang, and Guanghai Cui. 2017. Resilient routing overlay network construction with super-relay nodes. *KSII Transactions on Internet and Information Systems (TIIS)* 11, 4 (2017), 1911–1930. doi:10.3837/tiis.2017.04.005

[24] Van Tong, Sami Souihi, Hai Anh Tran, and Abdelhamid Mellouk. 2021. Machine learning based root cause analysis for SDN network. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6. doi:10.1109/GLOBECOM46510.2021.9685185

[25] Hui Yang, Xudong Zhao, Qiuyan Yao, Ao Yu, Jie Zhang, and Yuefeng Ji. 2020. Accurate fault location using deep neural evolution network in cloud data center interconnection. *IEEE Transactions on Cloud Computing* 10, 2 (2020), 1402–1412. doi:10.1109/TCC.2020.2974466

[26] Haijun Yang, Qinghua Zheng, Minqiang Li, and Yuzhong Sun. 2015. How to avoid herd behavior: A stochastic multi-choice scheduling algorithm and parameters analysis in grid scheduling. *International Journal of Information Technology & Decision Making* 14, 02 (2015), 287–315. doi:10.1142/S0219622014500734

[27] Jin Y Yen. 1971. Finding the k shortest loopless paths in a network. *management Science* 17, 11 (1971), 712–716. doi:10.1287/mnsc.17.11.712

[28] Zhi-Hua Zhou. 2018. A brief introduction to weakly supervised learning. *National science review* 5, 1 (2018), 44–53. doi:10.1093/nsr/nwx106

[29] Yaping Zhu, Benjamin Helsley, Jennifer Rexford, Aspi Siganporia, and Sridhar Srinivasan. 2012. LatLong: Diagnosing wide-area latency changes for CDNs. *IEEE Transactions on Network and Service Management* 9, 3 (2012), 333–345. doi:10.1109/TNSM.2012.070412.110180

[30] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491. doi:10.1145/2829988.2787483