M Ű E G Y E T E M 1 7 8 2

MSc Thesis Task Description

Bálint Bicski
candidate for MSc degree in Computer Engineering

Service Degradation Detection Using Early Network Traffic Flow Characteristics

Ensuring the Quality of Service (QoS) for a given network service is crucial for providing a satisfactory end-user experience. Degradation of QoS parameters such as latency, throughput, jitter, and packet drop can substantially impact the end-user experience and lead to service degradation. Such degradation may arise from various factors, including network congestion, hardware or software faults, or cyber-attacks. Real-time monitoring and identification of network service degradation are essential for service providers to assure reliable performance and maintain customer satisfaction. The objective of this thesis is to develop a methodology that effectively identifies and predicts network service degradation using early flow statistical characteristics.

Tasks to be performed by the student include:

- Conduct a literature review on the role of QoS parameters in the context of network performance and their impact on end-user experience.
- Investigate network traffic flows and analyze early statistical characteristics that can be used to predict service quality degradation.
- Propose and implement a novel methodology for forecasting service degradation based on the identified early flow statistical features, considering factors such as real-time applicability, scalability, and accuracy.
- Validate the performance of the proposed methodology using a dataset of network traffic flows.
- Discuss potential limitations of the developed methodology and propose possible improvements or future research directions.
- Compile the findings, methodology, and validation results into a well-structured thesis document, adhering to the required format and citation style.

Supervisor at the department: Dr. Adrian Pekar, senior research fellow

Budapest, 28 March 2023

/ Dr. Sándor Imre /
professor
head of department

Supervisors' proposals:
Supervisor at the department: ☐Acceptable, ☐Not acceptable, date: signature:

MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Service Degradation Detection Using Early Network Traffic Flow Characteristics

Master's Thesis

Author
Bálint Bicski

Advisor
Dr. Adrián Pekár

May, 2024

Contents

Kivonat

Ez a dolgozat átfogó tanulmányt mutat be a szolgáltatásromlási (Service Degradation – SD) események azonosításáról és előrejelzéséről hálózati környezetekben, amelyek mind egy egyetemi kollégium LAN adatait, mind a hálózati forgalomban található korai folyam jellemzőket magukban foglalják. A kutatás első része statisztikai technikákat, például interkvartilis terjedelem (Inter Quartile Range – IQR) és Z-metrika elemzéseket alkalmaz, a hálózati teljesítmény jelentős eltéréseinek felismerése érdekében, ezalatt különösen értve a szélsőséges késleltetéseket és jittert, amelyek potenciális SD-t jeleznek. Ezt a módszert különböző környezetekben alaposan validáltam, minimális eltéréseket tapasztalva, és megerősítve annak következetességét és megbízhatóságát. A tesztek azt sugallják, hogy a módszer alkalmazható mind a lakossági, mind a vállalati hálózatokban, így rendszerezett módszertani alkalmazásokat és értékes annotált adatkészletet nyújtva a terület kutatói számára.

A kutatás második része egy új módszert mutat be az SD előrejelzésére hálózati folyamatokban korai flow jellemzők (early flow features) elemzésével, különösen a csomagok közötti érkezési időre (PIAT) fókuszálva. A módszer a hálózati folyamatok megfigyelhető (Observable – O) szegmenseire összpontosít, hogy következtetéseket vonjon le a nem megfigyelhető (Non Observable – NO) szegmensek viselkedéséről. Egy átfogó analízissel azonosítottam az optimális O/NO szétválási küszöbértéket 10 csomagnál, amely egyensúlyt teremt az előrejelzési pontosság és az erőforrás-felhasználás között. A tesztelt modellek közül az XGBoost felülmúlta a többit, 0,74-es *F_1-score*-ral, míg a *balanced accuracy*-je 0,84 és *AUROC* értéke 0,97 volt. A regressziós elemzés kimutatta, hogy az XGBoost (és a többi vizsgált modell) kevésbé pontos az SD események számának, hosszának és a leghosszabb SD események indexeinek előrejelzésében.

Összességében ez a dolgozat kiemeli e módszerek megvalósíthatóságát és hatékonyságát az SD események észlelésére és előrejelzésére, robusztus keretrendszert biztosítva a hálózati adminisztrátorok számára a potenciális szolgáltatásromlás előzetes kezeléséhez. A jövőbeli munkának az adaptív küszöbértékek finomítására, a folyamok közötti információk feltárására és ezeknek a módszereknek a különböző hálózati környezetekben történő validálására érdemes összpontosítani a robusztusság és alkalmazhatóság növelése érdekében.

Abstract

This thesis presents a comprehensive study on identifying and predicting Service Degradation (SD) events in network environments, encompassing both LAN data from a university dormitory and early flow features in network traffic. The first part of the research leverages statistical techniques, such as Interquartile Range (IQR) and Z-score analyses, to detect significant deviations in network performance, specifically extreme delays and jitter, indicating potential SD. This methodology was rigorously validated across various settings, demonstrating minimal deviations and reinforcing its consistency and reliability. Initial tests suggest its potential applicability in both residential and enterprise networks, contributing systematic methodological applications and a valuable annotated dataset to the field.

The second part of the research introduces a novel method for predicting SD in network flows by analyzing early flow features, particularly Packet Inter-Arrival Time (PIAT) values. By focusing on the observable (O) segments of network flows, the method infers the behavior of non-observable (NO) segments. Comprehensive evaluation identified an optimal O/NO split threshold of 10 packets, balancing prediction accuracy and resource utilization. Among the models tested, XGBoost outperformed others, achieving an F_1-score of 0.74, a balanced accuracy of 0.84, and an AUROC of 0.97. The regression analysis revealed that XGBoost (and the other models) have limited accuracy in predicting SD count, length, and indices of the longest SD events.

Overall, this thesis underscores the feasibility and effectiveness of these methodologies for SD event detection and prediction, providing a robust framework for network administrators to preemptively address potential service degradation. Future work should focus on refining adaptive thresholds, exploring inter-flow information, and validating these methods in diverse network environments to enhance their robustness and applicability.

Chapter 1

Introduction

In the era of relentless digitization, computer networks have become critical to our everyday digital activities, underpinning everything from streaming high-definition videos to facilitating real-time multiplayer games. As the demand for uninterrupted, high-quality digital experiences grows, so does the need for resilient and high-performing networks that can consistently meet these expectations.

However, service degradation (SD) poses a substantial threat to these expectations. SD, characterized by the deterioration of network performance, leads to suboptimal user experiences due to factors such as network congestion, inefficiencies in data routing, or external interferences [18, 31, 5]. The effects are noticeable: users may experience increased latency, buffering in video streams, or even complete service outages. In essence, SD undermines the promise of a seamless digital experience.

This thesis aims to study two main problems: *(i)* It examines network behavior in a measured network flow dataset, with the intention of identifying the flow features and behavioral patterns that are indicative of SD. And *(ii)* it explores the constrained environment of residential network devices to harness a small portion of the forwarded data, an analyze whether it can be utilized for predicting behavior in later parts of the traffic.

SD Pattern Analysis Despite various previous contributions, gaps remain in accurately diagnosing the signs of SD in computer networks. Building upon these foundational studies, this research employs Packet Inter-Arrival Time (PIAT) analysis in a university dormitory network setting—particularly susceptible to SD—to focus on latency as a principal indicator of SD. My analysis specifically targets TCP flows and LAN segments, where PIAT, analyzed through Z-score and Interquartile Range (IQR) methods, provides critical insights into response latency without the confounding effects of end-device delays.

I present a robust methodology for detecting SD events in network flows. By applying empirical heuristics across all application categories, I effectively identify and label these events, thereby enhancing the understanding of SD within networked environments. Corresponding digital artifacts of my methodology are provided as supplementary materials to support the replicability of the analysis. The labeled dataset of network flows with identified SD events is also available at a digital source ([13, 12]), serving as a valuable resource for further research into network service degradation.

The methodology has proven robust when tested across various network environments, suggesting its applicability beyond the initial university dormitory setting. The identified

SD events correlate strongly with potential impacts on the Quality of Experience (QoE), suggesting areas for network improvement and further research.

Network Behavior Prediction in Residential Environments To proactively counteract SD, it is crucial to gain insights into network flows through comprehensive monitoring. Yet, network devices, which serve as the gatekeepers of residential networks, often have limited computational capabilities. While they are proficient at forwarding traffic, they struggle with the intensive monitoring of every flow due to hardware limitations. Instead of exhaustively monitoring every packet in a data flow, a common workaround is to observe each flow up to a predetermined packet threshold. Once this threshold is reached, flow management transitions to hardware accelerators (HAs), which handle high volumes of data efficiently but lack the capability for detailed, packet-level monitoring. Consequently, this approach effectively splits flows into observable and non-observable parts, leading to the loss of detailed insights into the flow's behavior beyond the threshold.

This situation leads to my primary research question: *In the face of these constraints, how can we leverage the observable segments of network flows to deduce the status of the non-observable segments, thereby identifying and possibly predicting SD in residential networks?* The core of my approach is to derive patterns from the observable segments, which can then be extrapolated to make informed assumptions about the non-observable segments and, by extension, the network's overall health.

This study introduces a method called *intra-flow service degradation detection*, which uses early flow features to predict SD in network flows after they transition from an observable (O) state to a non-observable (NO) state. By capturing information from the initial part of a flow, I aim to infer the behavior and characteristics of the flow after it moves to a fast-processing but less observable state.

My experimental design focused on evaluating the predictive power of observable parts of network flows to predict various metrics in the non-observable parts. I targeted five distinct objectives: detecting the presence of SD events, predicting the count, length, and indices of the longest SD events. I compared multiple models, including simple heuristics and advanced techniques such as Logistic Regression, XGBoost, and Multi-Layer Perceptron (MLP). This comprehensive assessment aimed to identify the most effective models for each objective.

The classification analysis I apply reveals that the XGBoost model, evaluated at an O/NO split threshold of 10, excels in predicting non-observable SD events. This model outperforms simpler heuristics and other sophisticated models by offering a balanced performance with high precision and good recall. The XGBoost model's robustness and efficiency make it the optimal choice for this prediction task, highlighting the importance of incorporating comprehensive early flow features. The chosen threshold of 10 strikes an effective balance, providing sufficient information for accurate predictions without unnecessary complexity.

My regression analysis indicated that the XGBoost model performed best for predicting non-observable SD events, especially at O/NO split thresholds of 5 and 20. This model consistently provided more accurate predictions for SD count, length, and indices of the longest SD events with the heuristic models showing slightly less effectiveness. However, meanwhile the models were able to identify whether a flow had SD events after the O/NO split at all, for flows with SD events in the NO parts, all models, including XGBoost, struggled to predict more granular information accurately. This highlights the challenge of predicting detailed SD characteristics in non-observable parts and underscores the need for robust models and sufficient information from observable parts.

This research demonstrates the feasibility and effectiveness of using early flow features to predict SD in network flows after they become non-observable. Ultimately, this work offers a blueprint for early detection and mitigation of service degradation issues, ensuring a more resilient and high-performing network. Through proactive monitoring and analysis, I aim to prevent potential disruptions before they impact the end user, thereby delivering a seamless digital experience.

The source codes used for analysis and model evaluation are also provided as digital artifacts, alongside with the detailed results of the evaluated model metrics [14].

The rest of this thesis is organized as follows: *Chapter 2* explores pieces of related work focused on the analysis of SD in various domains. *Chapter 3* presents the methodology of network flow measurement and feature generation, introduces the dataset that is gathered using this technique and performs descriptive analysis on the data. In *Chapter 4*, I explore the concept of SD in network traffic, providing the necessary background, and execute a comprehensive statistical analysis to select LAN side delays from the collected flows and identify patterns that indicate SD in the the flows. Following this, I cross-validate on data collected at different times and locations. *Chapter 5* describes the limitations of the home network environment, exploring the operation of residential routers. Along this line I propose a methodology of separation of the network data into an O and NO part, and investigate the statistical behavior of the delays and SD events in each part along different splits, and evaluate the predictive performance of the O part information for forecasting NO behavior. Finally, *Chapter 6* concludes my work.

Chapter 2

Related Work

Table 2.1: Summary of Related Works and Proposed Work

Reference	Methodology	Network Type	Key Findings	Year
Bremler-Barr et al. [4]	Degradation Analysis	Internet End-to-End	Identified RTT deviation	2003
Abdelkefi et al. [1]	PCP of delay and loss signals	Internet End-to-End	Scalable and cost-effective model for Internet path service quality assessment	2014
Xian et al. [35]	Prediction-based Algorithm	Optical Data Center	Matched theoretical prediction with actual traffic, mitigated network congestion	2016
Hou et al. [17]	Forecasting Algorithm	Optical Data Network	Mitigated network congestion	2019
Hardegen et al. [15]	DNN	Campus Network	Predicted traffic flow characteristics	2020
Marra and Corman [21]	Delay to Disruption Analysis	Public Transportation Networks	Analyzed real disturbances to identify disruptions	2020
Cortellessa and Traini [9]	Latency Degradation Detection	Microservice-based Applications	Detected artificially injected latency degradation	2020
Traini and Cortellessa [30]	LDP Detection	Service-based Systems	Diagnosed performance issues	2023
Wu et al. [33]	Real-Time Packet Loss Monitoring	-	Emphasized real-time passive packet loss detection	2023
Pons et al. [27]	GIPS metric	Public Cloud VMs	Detected inter-VM interference and estimate performance degradation, identify degraded VMs	2023
Amaral et al. [2]	Heuristic model for analyzing accumulated latency	Cloud Gaming	Anticipated visual degradations before they occur, allowing proactive mitigation	2023
Li et al. [19]	Adaptive Bitrate Algorithm	Live Video Streaming	Stochastic adaptation of bitrate based on network conditions and client states to improve QoE	2023
Zhu et al. [36]	Degradation Aware User Allocation	Edge Computing	Optimized resource usage to increase the number of users served by introducing controled SD	2024
Proposed Work	PIAT Analysis	Residential LAN	Detected LAN-side service degradation and predicted degradation in the non-observable part of the traffic	-

The endeavor to discern network behavior, particularly in foreseeing service degradation, has been an ongoing pursuit in the domain of network monitoring and optimization, though only a limited number of studies directly intersect with our specific domain of interest. An early study by Bremler-Barr et al. [4] from 2003 delved into the predictability of end-to-end Internet service degradations by analyzing deviations in round-trip time (RTT) between clients and servers. Their work aimed at understanding the patterns and predictability of service degradations, providing insights into how network performance can be analyzed and predicted over time.

Abdelkefi et al. [1] present a method to assess Internet path service quality using end-to-end delay and loss measurements, called SCI (Service-quality Characterization of Internet-path). SCI constructs performance signals from these measurements, including aggregate delay, average delay, and aggregate loss signals. Abrupt changes in these signals are detected using Principal Component Pursuit (PCP), and the detected changes, along with loss information, are mapped to service-level events causing degradations and failures. The approach quantifies service quality through three metrics: availability, stability, and fatigue. The authors evaluate their work on a dataset collected from a real operational environment demonstrates that SCI effectively detects abrupt changes and accurately identifies service-level events, thereby confirming its feasibility for service quality assessment based solely on end-to-end measurements.

Hardegen et al. [15] employed Deep Neural Networks (DNN) to predict certain characteristics of traffic flows within a campus network. Their approach utilized DNN to estimate the throughput and duration of flows, offering a proactive means to understand network behavior before notable degradations occur. The results from their study illustrated a promising avenue for using machine learning techniques in predicting network flow characteristics, albeit in settings with ample computational resources. A real-time packet loss monitoring system proposed by Wu et al. [33] addressed the issue of packet loss leading to network quality of service degradation. Though the detailed methodology and results were not available, the work emphasizes real-time passive packet loss detection for estimating network service quality.

Traini and Cortellessa [30] explored latency degradation patterns in service-based systems using a concept called Latency Degradation Patterns (LDPs) to diagnose performance issues. They integrated LDP detection in the software development process, demonstrating how recognizing patterns in latency could be instrumental in diagnosing performance issues in service-based systems. In a similar environment, Cortellessa and Traini [9] delved into detecting latency degradation patterns in microservice-based applications, identifying clusters of requests subject to latency degradation. Through a case study, the effectiveness of their approach in detecting artificially injected latency degradation patterns was demonstrated.

Pons et al. [27] introduce Cloud White, an approach designed to detect inter-VM interference and estimate performance degradation in public cloud environments running multiple latency-critical virtual machines (VMs). The presented work addresses the challenge of identifying victim VMs – i.e. those experiencing performance degradation due to resource contention – and estimating their performance loss using multi-variable regression models. The approach leverages the Giga Instructions Per Second (GIPS) metric to discern victim VMs from inflicting VMs, which increase their load and cause interference. Cloud White dynamically estimates the performance degradation, focusing on the 95th percentile tail latency, ensuring compliance with Quality of Service (QoS) requirements. Experimental results demonstrate that the approach achieves a low prediction error, allowing cloud providers to optimize server utilization while mitigating the need for overprovisioning.

The scope of SD research also extends to the transporation sector. Its implications are explored in public transportation systems in a study by Marra and Corman [21]. The authors investigated the transition from delay to disruption and its impact on service degradation, analyzing real disturbances in Public Transportation Networks to identify disruptions with different characteristics. This work provides a different perspective on understanding the relationship between delays, disruptions, and service degradation in networks. In the automotive industry, with a focus on synchronized systems, the detection of time-delay attacks and their impact on traffic latency was studied by Luo et al. [20]. While the detailed methodology was not available, the research hinted at detecting and locating time-delay attacks in Time-Sensitive Networking (TSN), thus addressing latency variations in a specific networking context.

Recent studies have particularly highlighted latency as a reliable indicator of video quality degradation in online gaming and video streaming. Amaral et al. [2] investigate the impact of network impairments on video quality in cloud gaming and introduces an algorithm to predict real-time visual degradations. Cloud gaming, which runs games remotely in the cloud and streams them to users, faces challenges due to its heavy reliance on network performance. The research identifies increasing accumulated latency in a moving window as a strong predictor of packet loss, which leads to visual degradation. Utilizing this insight, the developed algorithm anticipates visual degradations before they occur, enhancing the gaming experience by allowing proactive measures to be taken. Extensive testing across various video game scenarios demonstrates the algorithm's effectiveness in predicting and mitigating visual quality issues, thus improving the reliability and playability of cloud gaming platforms. Following similar veins, the study of Li et al. [19] introduces Fleet, an adaptive bit rate algorithm designed to enhance user QoE in low-latency live video streaming. Fleet addresses the dual challenges of optimizing QoE under dynamic mobile network conditions and ensuring low latency with minimal visual quality degradation. It employs a stochastic model predictive controller that integrates network conditions and client states for bitrate adaptation. Key components of Fleet include an HTTP chunk-level bandwidth measurement algorithm to tackle the idle period problem, a throughput probability predictor for capturing mobile network uncertainties, and a triple threshold playback speed controller for latency management. A comprehensive evaluation confirms Fleet's effectiveness in improving live video streaming experiences, even in fluctuating network environments.

On a different note, some perspectives in the literature treat SD not merely as a technical challenge but as a strategic resource management tactic. These studies suggest that SD can be intentionally implemented to strategically degrade service resources, aiming to maximize profitability even if it requires deliberate reductions in Quality of Service (QoS) to enhance overall profitability.

Xian et al. [35] proposed an alogrithm for predictive resource allocation in Optical Data Center Networks. Service degradation is utilized in this approach to manage network overloads by predictively reducing the Quality of Service (QoS) in a controlled manner when the forecast traffic load is high, thus preventing service denial. Simulation results indicate that their approach effectively increases provider profit and reduces network congestion by lowering the blocking rate. Similarly, Hou et al. [17] presented a forecasting algorithm to support service degradability in optical data networks. Their algorithm aimed at maximizing service provider profit by mitigating network congestion, a crucial aspect in maintaining network performance. The results demonstrated a notable reduction in network congestion, thus maximizing profit. Zhu et al. [36] address the high service demand problem in edge computing. In this environment heterogeneous and resource-constrained edge servers struggle to meet the increasing service demands, especially during peak periods. Instead of focusing fulfilling prescribed service requirements totally that often result in significant user loss, the authors explore the concept of controlled service degradation. By allowing a certain degree of service requirement violations, more users can be accommodated, thus optimizing resource usage. However, to balance this, users receive appropriate compensation for any degradation, ensuring overall satisfaction without incurring excessive costs that could reduce service profit.

Contrastingly, our work navigates the intricacies of residential LAN environments, where resource constraints are prevalent. Our approach, based on PIAT analysis up to a predetermined packet threshold, seeks to provide a balanced approach between resource conservation and effective network monitoring. In line with the presented related works, we also distinguish latency as a key metric for service degradation identification and use statistical methods to pinpoint significant outliers that indicate SD. Our work also emphasizes practical implications in hardware-constrained environments, targeting the internal segment of networks to provide early indications of potential SD. Our preliminary findings underscore the potential of this approach in paving the way for refined methods of monitoring service degradation in such constrained environments, thus contributing a novel perspective to the domain of network monitoring and optimization.

Chapter 3

Network Flow Data Collection

In this chapter I explore the process of network flow data collection and present the collected data using this approach. Section 3.1 introduces the main concepts for flow measurement, including the definition of a network flow, the pipeline used for assigning packets to flows, and the generation of various flow features. It also presents the flow measurement framework, NFStream, that I utilized in this thesis for the purposes of data collection. Section 3.2 describes the measured network traffic dataset presenting the parameter settings used for data collection. The section provides a comprehensive overview of the data distribution, both in terms of the temporal characteristics and flow feature characteristics. The analysis is also found complete with codes in a Jupyter Notebook file at [13].

3.1 Network Flow Measurement

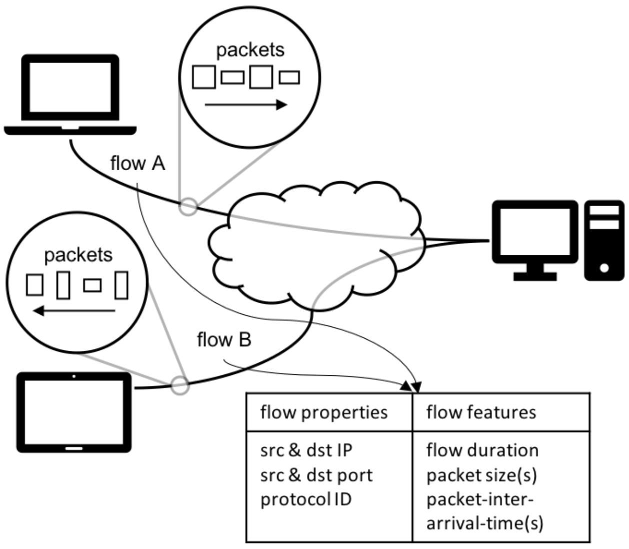


Figure 3.1: The relationship between packet streams, flows, flow properties, and flow features.

A flow represents a collection of network packets observed at a particular point over a defined timeframe. Packets within the same flow share common *properties*, typically identified by a five-tuple. This five-tuple generally includes the source and destination IP addresses, protocol identifier, and the source and destination ports. For a precise definition of flow, refer to [32]. The calculation of *flow features* relies on the packets associated with the flow. Measurable features can include the sizes of the packets (referred to as packet size or PS) and the intervals between observing consecutive packets in the same flow (referred to as packet-inter-arrival-time or PIAT). The connection between flows, their properties, and their features is illustrated in Figure 3.1.

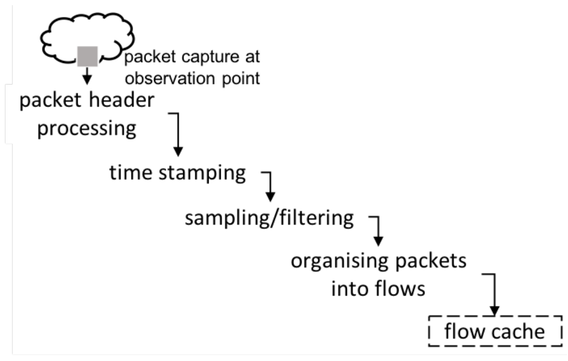


Figure 3.2: Packet processing phases from capture to flow entries.

The process from packet capture to creating flow entries is detailed in Figure 3.2. Initially, packets are timestamped (necessary for computing time-related features like flow start, end, and duration) and then undergo packet selection. This optional step, often used to manage resource constraints (e.g., computational or storage), reduces the number of packets to be processed. Packet selection can involve sampling (choosing a subset of packets) or filtering (focusing on specific endpoints or IP ranges). Unselected packets are discarded. The selected packets are then organized into flow entries.

Flows with packets moving from one endpoint to another are termed *unidirectional flows* (uniFlows). However, because network paths can be asymmetric, flow features may differ statistically depending on the direction. Therefore, directionality is crucial in traffic measurement. For flows with packets traveling in both directions between endpoints (in essence two unidirectional flows in opposite directions), *bidirectional flows* (biFlows) are considered. The distinction between uniFlows and biFlows is shown in Figure 3.3.

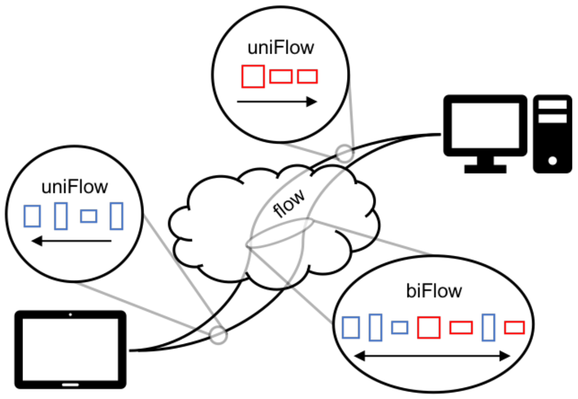


Figure 3.3: The difference between unidirectional flows and bidirectional flows.

In practical applications, flow entries are managed in a *flow cache*. Managing flow entries involves creating, updating, and identifying expired entries. Figure 3.4 illustrates the analysis and comparison of each captured packet against flow cache entries. A new entry is created when a packet does not match any existing flow in the cache. Mapping new packets to flow records is challenging, especially with a large number of flow records in the cache. To optimize lookup time, each entry is stored with a *flowID*, a hash based on the five-tuple. For each new packet, a *packetFlowID* is computed and compared to existing flowIDs in the cache. If the IDs match, the packet is used to update the features of that specific flow. A backward flow ID (*bwdFlowID*) is also stored for each flow entry, computed with swapped source and destination IP addresses and port numbers, to match packets in the reverse direction for bidirectional flows. If an arriving packet matches this, apart from the general, direction–agnostic flow features, only the features corresponding to the backward direction are updated.

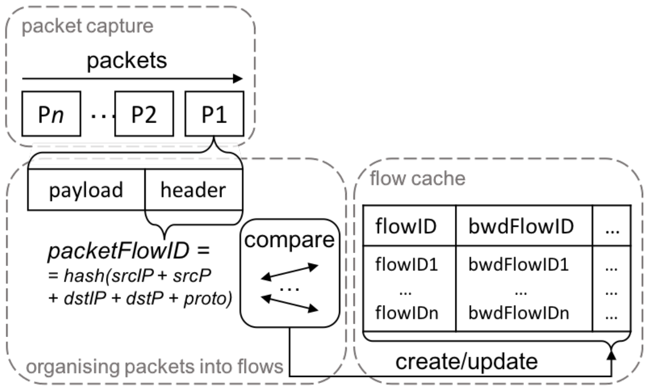


Figure 3.4: Organising the captured packets into flows.

There are a large amount of flow features that can be captured or computed in flow measurement [23, 25]. Many of these include statistical transformation of aggregated packet-level features, like the mean, minimum, maximum or standard deviation of packet sizes or PIATs. Others are counters of specific packet characteristics, like TCP packet flags. Flow level features are also often recorded, like the packet count or total duration of the flow. The former can be derived by a simple counter, while the latter is the difference between the arrival time of the last packet and the first packet in the flow. All the aforementioned features can be calculated for biFlows taking into consideration all the packets together, or the two directions separately. Other features that require Deep Packet Inspection (DPI), for instance information regarding the underlying application or service name of the flow. By inspecting the IP and MAC addresses further service or device specific information can be gathered, like the device vendor from the Organizational Unique Identifier (OUI) in the MAC, or the Autonomous System Number (ASN) or Server Name Indicator (SNI) corresponding to the IP addresses and hostnames.

Tracking active flows and detecting their expiration are essential for managing flow entries in the flow cache. Flows can expire for various reasons. One reason is if no packets from the flow have been observed for a specified period, known as *passive* or *idle timeout*. Passive expiration prevents incorrectly identifying different flows as the same. For example, a host initiating multiple new TCP connections to another host might reuse the same source port after exhausting the range, causing confusion if not expired appropriately. Another reason for regular expiration is long-lasting flows. For real-time network management, it is impractical to store such flow entries for extended periods. For instance, a network administrator might need to know about a lengthy file download promptly rather than hours later. Thus, flows are regularly expired even with continuous packet flow, termed as *active timeout*. Additionally, flows naturally expire when a TCP packet with FIN or

RST flags is observed. Note that natural expiration requires understanding the transport protocol's signaling semantics to avoid missing teardown or out-of-sequence packets. Other less common expirations include emergency expiration (to prevent system crashes when the flow cache is full) and cache flushes (required during system reconfigurations affecting flow entries). For a more comprehensive discussion on flow measurement, refer to [16].

3.1.1 Network Data Analysis Framework

For flow measurement and feature computation I employed the open-source NFStream [3] framework, enabling traffic measurement at high network throughput. The utilization of this framework ensures the reproducibility of my work. The framework captures numerous flow characteristics, such as IP, TCP, and UDP packet addresses, and is also suitable for collecting early flow characteristics. Furthermore, it comprises a wide array of computational statistical analysis-centric features, making it suitable for producing in-depth analytics. NFStream also employs the nDPI library [10] for Layer 7 visibility through DPI, extracting the application type carried by the flow and other specific pieces of data.

Additionally, the solution enables users to implement modular plugins for network monitoring, integrating custom logic into the observation process and facilitating the development of unique methods without the need to create new network flow analysis software. The generated modules are portable and do not necessitate hardware configuration requirements, owing to the robustness of NFStream.

3.1.2 Early Flow Statistics

Early flow features are observation-based analytical indicators that are calculated from packet characteristics at early stages of the flow. Opposed to conventional flow features (like IP address or protocol) they are susceptible to temporal service circumstances making them ideal indicators of early flow behavior. Using the `splt_analysis` parameter during the execution of the NFStream flow generation, the number of packets reserved for early flow feature extraction can be set.

In my work, the focus is analyzing the the flow behavior in depth at the early stages. To this end, I will employ three early flow features, which are as follows:

(i) NFStream compiles the packet directions of a given flow in the **splt_direction** list. Three distinct values are identified in terms of direction. A 0 marks a packet traveling from source to destination while a 1 stands for the opposite direction. In the case of -1, no packet was sent in this state of the flow (e.g. due to the flow terminating early).

(ii) The framework stores the packet size values in the **splt_ps** list, with the packet size measured in bytes. It is also possible that no packet was sent; in this case, we cannot interpret the size of the missing packet, and the framework marks the packet absence with -1.

(iii) Finally, the times elapsed since the arrival of the last packet, i.e. the PIATs are stored in the **splt_piat_ms** list recorded in milliseconds. For the first packet in the flow, this value always takes on a 0 value, and for the missing packet, as with the previous cases, it is marked with -1.

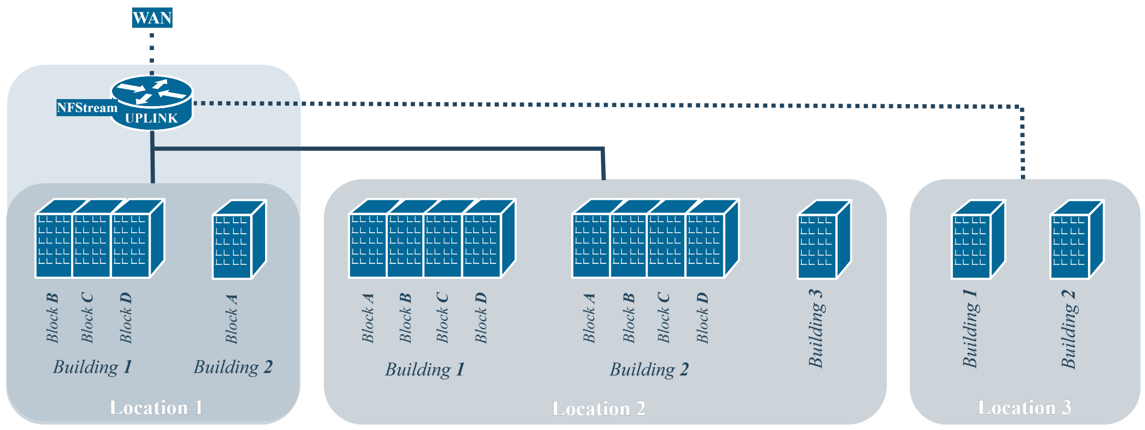


Figure 3.5: High-level topology of the measured network.

3.2 Dataset

The dataset utilized in my study comprises network traffic data from a university dormitory network. The topology of the network, as shown in Figure 3.5, provides a high-level overview, abstracting specific details like the nature of network connections and internal building topology into simplified representations. Three primary locations within the dormitory network are depicted, each based on the physical grouping of buildings. *Location 1*, positioned on university premises, serves as the uplink router site where traffic was captured via a Switched Port Analyzer. *Location 2* is situated a few kilometers away on the city outskirts, and *Location 3* is located in a different town, connected through multiple hops as indicated by a dotted line in the figure.

Table 3.1: Characteristics of the Captured Network Flow Data

	Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
Flow Count	3,185,436	2,513,156	3,092,647	2,703,948	2,865,281
Start of the Measurement	2024-01-15 19:24:40	2024-01-16 19:34:09	2024-01-17 19:26:35	2024-01-18 19:20:36	2024-01-19 19:20:35
End of the Measurement	2024-01-15 20:18:29	2024-01-16 20:16:14	2024-01-17 20:18:01	2024-01-18 20:15:01	2024-01-19 20:15:01
Measured Timespan	53.81 minutes	42.09 minutes	51.44 minutes	54.41 minutes	54.43 minutes
Min Flow Rate $[\frac{1}{s}]$	78	312	385	70	319
Max Flow Rate $[\frac{1}{s}]$	3469	3511	3766	3199	2651
Mean Flow Rate $[\frac{1}{s}]$	986.2	994.92	1001.83	827.91	877.04
Std. Dev. of Flow Rate $[\frac{1}{s}]$	247.48	270.75	241.2	228.51	322.27
Min Data Rate [MiBps]	0.0587	0.8091	1.2242	0.0807	0.3239
Max Data Rate [MiBps]	78,087	108,119	73,567	137,937	83,792
Mean Data Rate [MiBps]	243.49	307.21	254.69	244.64	204.68
Std. Dev. of Data Rate [MiBps]	1475.07	2322.29	1517.05	2541.15	1603.97
Flow Count *after filtering*	1,464,421	1,097,956	1,400,967	1,089,312	923,214
Flow Count *after LAN-side delay extraction*	1,356,999	1,012,526	1,282,952	1,002,830	845,631

3.2.1 Flow Measurement Configuration

For network flow measurement, I comfigured NFStream as follows:

- I filtered the traffic capture to IPv4 TCP traffic only; the rationale for this choice is explained in Section 4.1.4.

- Flow expiration settings were configured to terminate all flows after 2 minutes of inactivity following the last received packet (passive timeout), or after 30 minutes regardless of activity (active timeout).
- Packet size accounting was configured to include the IP header, but tunnel decoding was not enabled for this measurement.
- The nDPI library [10] was used to dissect up to 20 packets, which allowed us to identify application usage.
- I analyzed statistical features such as packet size, PIAT, and packets with various TCP flags. Statistical measures (minimum, maximum, mean, standard deviation) were calculated for traffic in both directions—source to destination and vice versa—and combined.
- Our custom NFPlugin (code available as part of the digital artifacts at [13]) managed the expiration of TCP flows based on their natural termination. Specifically, a flow is terminated after an ACK that follows two FIN packets and does not carry a FIN itself. Additionally, any flow that begins with a FIN or RST packet is also terminated, diverging from the standard TCP three-way handshake process.

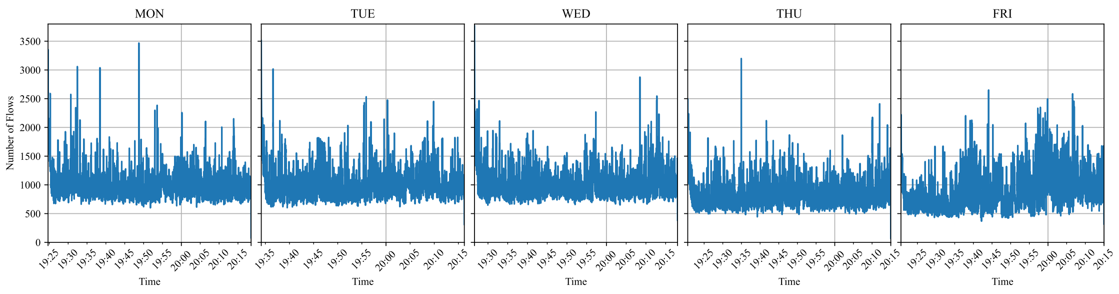
3.2.2 Temporal Characteristics

Network traffic was measured over the course of one week, specifically during the evening hours. Each session lasted approximately 40-50 minutes, depending on the volume of traffic captured that day. Table 3.1 details the specific measurement windows and durations for each day, as well as the daily flow counts and the minimum, maximum, mean, and standard deviation for flow arrival and data throughput rates.

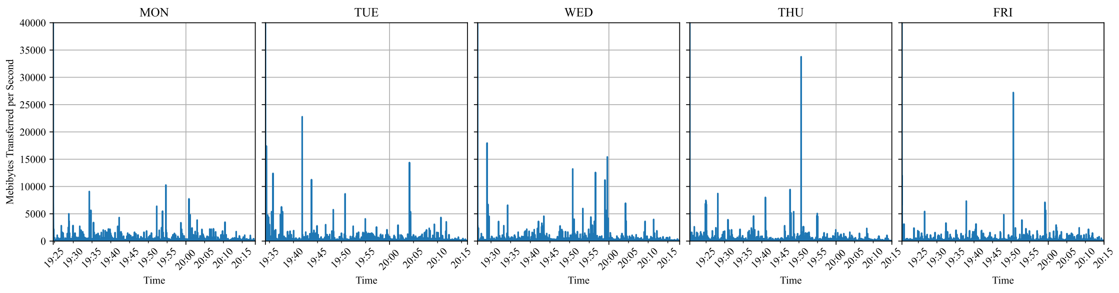
Figure 3.6a illustrates the daily flow arrival rate per second, showing considerable variability. Generally, the flow arrival rates averaged around 900-1000 flows per second, with a standard deviation of approximately 250. Notable spikes in the data, with peaks exceeding 3500 flows per second, indicate periods of intense activity or bursts of flow arrivals. While the rate frequently remained below 1500 flows per second between these peaks, lower burst arrival rates were observed on Wednesday and Friday, though the average rates were consistent with other days. This pattern suggests intermittent periods of heightened activity amid generally steadier or lower rates of flow arrivals. Factors contributing to these fluctuations could include typical network usage patterns, scheduled events such as database updates, or anomalies within the monitored network. These peaks may highlight potential instances of SD.

Figure 3.6b illustrates the data transfer rates observed during the measurement period. Generally, rates were relatively low, often remaining below 5,000 MiBps. However, notable exceptions include spikes that exceeded 30,000 MiBps, particularly around 19:50 on Thursday and again at the same time the following day, suggesting a common cause. These spikes were abrupt and short-lived, indicating brief periods of very high data transfer activity before returning to the baseline level. On average, data rates varied between 200-300 MiBps, with a standard deviation ranging from 1400 to 2400 MiBps. The distribution and magnitude of these spikes suggest irregular and potentially unpredictable bursts in data transmission, possibly due to activities such as scheduled data transfers, network backups, or streaming of high-definition media.

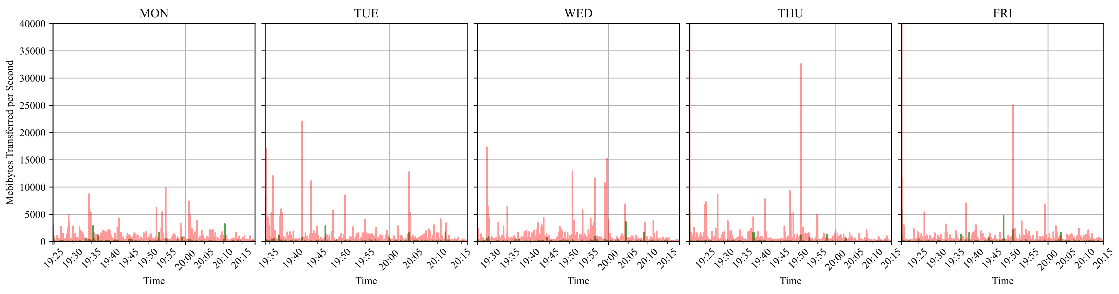
Further analysis in Figure 3.6c reveals that the significant spikes around 19:50 on both Thursday and Friday predominantly correspond to inbound traffic, with negligible out-



(a) Flow arrival rate $[\frac{1}{s}]$.



(b) Data rate [MiBps].



(c) Directional data rate [MiBps]. Green: upload (outbound) data Rate. Red: download (inbound) data rate

Figure 3.6: Daily distribution of temporal features during the week.

bound traffic. This pattern of high download activity with minimal uploads was consistent across all notable spikes, hinting at a heavy inbound data flow.

Interestingly, the flow arrival rate and the data rate do not appear closely correlated. While both metrics experience spikes, they occur at different times; the data rate peaks when the flow arrival rate is at normal levels and vice versa. This divergence could indicate scenarios where a high volume of transferred bytes accompanies a relatively low number of flows, potentially suggestive of an attack scenario.

Figure 3.7 presents the Empirical Cumulative Distribution Function (ECDF) plots for three key flow features: *packet count*, *flow size* (in bytes), and *flow duration* (in milliseconds). The packet count plot reveals that over 90% of flows consist of fewer than 100 packets, predominantly resulting in shorter flows. Notably, more than a quarter of all flows contain just a single packet, and only a small fraction of flows extend to millions of packets. The flow size ECDF exhibits a similar but more gradual trend; nearly 40% of the flows transfer less than 100 bytes in total, and over 90% contain no more than 10 KB of data. In contrast, the flow duration ECDF illustrates that while many flows are short

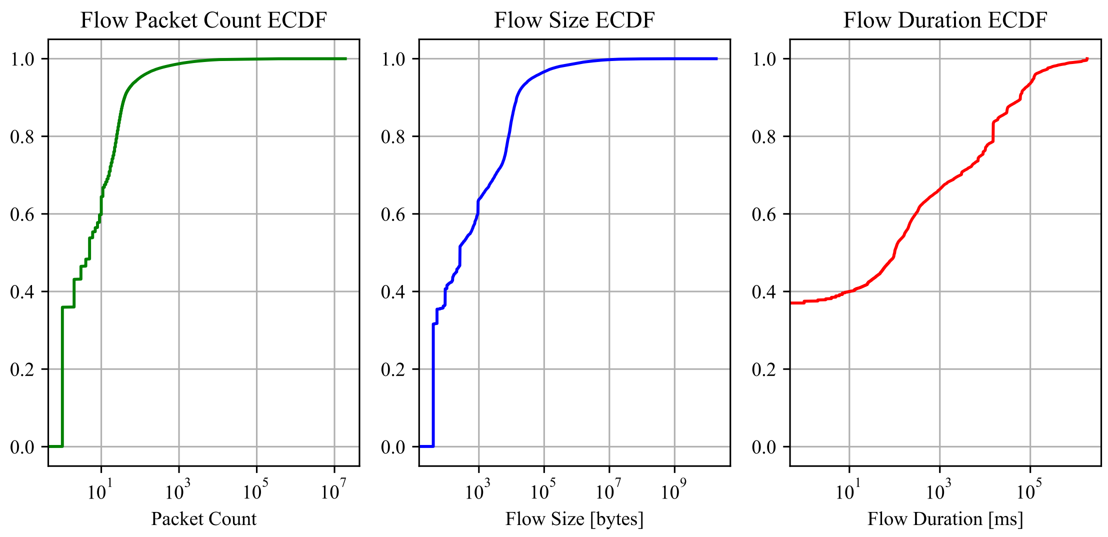


Figure 3.7: ECDF plots for packet count, flow size and flow duration.

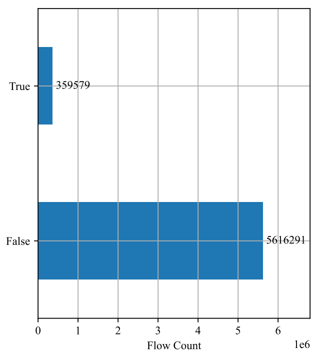
in terms of packet count and size, the duration of flows increases more gradually, with the longest flows exceeding 15 minutes.

Given my focus on analyzing delays that result in SD events in flows, particularly short flows, specifically those with fewer than two packets (indicating no response was received or sent), are excluded from my analysis. Additionally, flows were filtered based on the confidence level [10] determined by nDPI. The confidence level, indicated by a numeric value, reflects the certainty of the categorization; a higher number signifies greater confidence. Specifically, Level 6—achieved through Deep Packet Inspection rather than heuristic methods such as port-based approaches or correlations based on previous sessions—denotes the highest confidence and was the threshold for retaining data in this study. After applying these filters, approximately 40% of the flows from Monday to Thursday and 32% of Friday's traffic were retained for further analysis. The counts of flows retained post-filtering are detailed in Table 3.1.

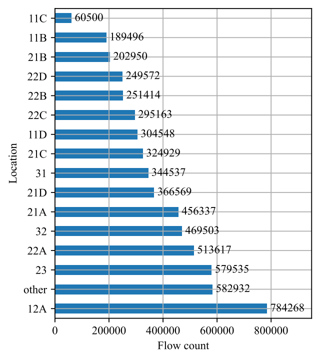
3.2.3 Feature Characteristics

In addition to temporal features, I analyzed the distribution of various categorical and numerical attributes, as shown in Figure 3.8. An examination of traffic directionality (Figure 3.8a) reveals that the overwhelming majority of flows originated from within the LAN and were directed towards WAN, with these non-reversed flows outnumbering reversed flows by an order of magnitude. Daily analyses show that while the volume of reversed flows remained constant throughout the week, non-reversed flows varied significantly. Closer inspection of the reversed flows indicated that they predominantly consisted of TLS and RDP traffic directed towards a specific host, characterized by highly consistent packet order and sizes. Notably, the majority of these reversed flows originated from a single external host, potentially representing deliberate connections to an open RDP port, though they may also reflect typical Internet background noise.

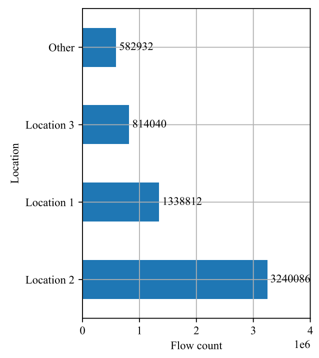
Figure 3.8b demonstrates that while the location with the highest traffic cardinality varied daily, 12A, 22A, 23, and 32 consistently reported high traffic volumes throughout the week. Here, the encoding convention follows a specific pattern: the first number represents the



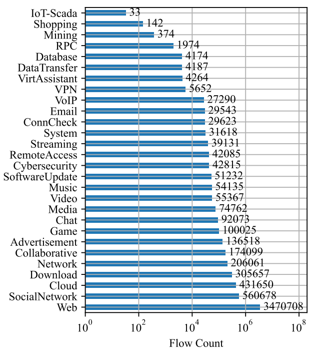
(a) Reverse traffic direction (*true*: WAN-to-LAN traffic, *false*: LAN-to-WAN traffic).



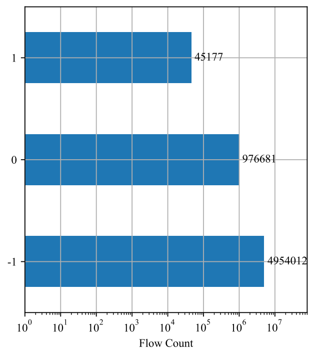
(b) Traffic source/destination building.



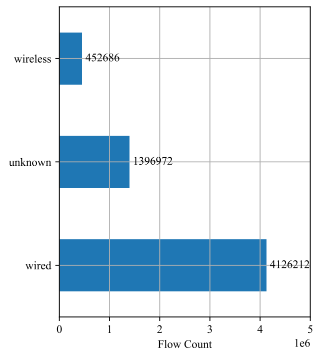
(c) Traffic source/destination location.



(d) Application category.



(e) Expiration ID (0: idle timeout, 1: active timeout, −1: natural TCP expiration).



(f) Connection type.

Figure 3.8: Cardinality distribution of dataset features.

location, the second denotes the building, and the last letter signifies the block, as outlined in Figure 3.5. For instance, 12A corresponds to *Location 1-Building 2-Block A*, while 32 corresponds to *Location 3-Building 2*. There was also a notable amount of traffic from locationally non-assigned sources. Conversely, 11C consistently recorded the lowest traffic, with only a few thousand flows each day.

When traffic is grouped by location (Figure 3.8c), *Location 2* emerges as the predominant source of flows, generating over 500,000 flows daily and exceeding 3 million flows in total for the week. Given that *Location 2* houses the largest number of buildings and students, such high traffic volumes are expected. The data shows that traffic from *Location 2* consistently surpassed that from all other locations combined throughout the week.

In my examination of the application category cardinality within the captured traffic, significant variances were observed. For instance, categories such as *Web* consistently showed high activity levels, with data points exceeding 500,000 daily and peaking at 800,000 on two occasions. In stark contrast, categories like *IoT-Scada*, *Shopping*, and *Mining* registered exceedingly low activity, with fewer than 50 data points each. Intermediate levels of activity were noted in other categories, ranging from several thousand to tens of thousands of data points. Figure 3.8d illustrates the cumulative distribution of application category cardinalities across the dataset. Additionally, the application names were also analyzed, providing a more granular view of the types of flows present, which could be pivotal for more detailed investigations.

Figure 3.8e presents the distribution of flow expiration types. Predominantly, flows were terminated through *natural TCP expiration* (indicated by -1 on the plot), which involves the standard FIN-ACK sequence in TCP connection terminations. Approximately 200,000 flows daily expired due to *idle timeout* (marked by 0 on the plot), occurring when no packets were received after two minutes of inactivity. *Active timeout*, which forcibly ends flows after 30 minutes regardless of activity, was a rare occurrence, affecting only a few thousand flows each day (indicated by 1 on the plot).

Finally, the distribution of connection types, as detailed in Figure 3.8f, reveals that the majority of traffic originated from wired connections, accounting for over 4 million flows. Notably, around 500,000 flows were initiated by devices recognized as wireless, while a substantial number of flows lacked specific connection type information.

3.2.4 Dataset Availability

In the spirit of fostering reproducibility and encouraging open scientific collaboration, this dataset is made publicly available [13].

To comply with GDPR, IP addresses and MAC addresses, which could be used to identify specific students, were anonymized using the *blake2b* algorithm [28]. Nonetheless, original IP range data was retained to annotate each flow with its location within the dormitory network. This setup not only preserves privacy but also provides crucial contextual information about the flow origins and destinations. Additionally, I distinguished whether the traffic was initiated by LAN hosts or directed towards the LAN, and identified whether the traffic originated wirelessly or via a wired connection, using the applicable addressing policy. This detailed dataset facilitates a deeper understanding of network dynamics and supports robust analysis of SD and other network issues.

Furthermore, to aid in the transparency and reproducibility of my research, I am also sharing the Jupyter Notebook file [13] used for analyzing the dataset. By providing these resources, I aim to support ongoing research in network flow analysis and contribute to the broader advancement of knowledge in this field.

Chapter 4

Unveiling Latency-Induced Service Degradation

This chapter explores the concept of SD on the data collected in Chapter 3, focusing on the extraction of true delays from the data and the analysis of this and other characteristics for SD identification. Source material used in the analysis including the source codes and the dataset complete with SD events are available as digital artifacts at [13] and [12].

Section 4.1 introduces foundational principles relevant to my approach, including the manifestation of SD, latency considerations, analysis of LAN and WAN segments, and the use of PIAT as the key metric, along with my data filtering strategies. Section 4.2 delves into my methodological approach, leveraging IQR and Z-Score methods to detect both singular and prolonged SD events. Section 4.3 explores the robustness and applicability of my methods across various settings. The chapter concludes with Section 4.4, summarizing my findings and their implications for network service management.

4.1 Foundational Concepts and Preparatory Considerations

4.1.1 Service Degradation in Networks

Service degradation, at its core, is the deterioration of network performance over time [6]. While seemingly straightforward, this deterioration manifests in varied ways, from intermittent connectivity drops to prolonged periods of sub-optimal throughput. The user's experience, thus, shifts from a seamless digital interaction to one fraught with delays, interruptions, and inefficiencies.

4.1.1.1 Causes of Service Degradation

Several factors are known to precipitate service degradation:

- **Network Congestion**: The most common culprit, congestion occurs when the demand surpasses the available bandwidth. This mismatch often results in packet loss, latency, and reduced throughput [18].
- **Hardware Failures**: Networks rely on a plethora of hardware components. Failures, be it due to wear and tear, manufacturing defects, or external factors, can degrade service.

- **Software Glitches**: Firmware or software running on network devices can have bugs or can be inefficiently optimized, affecting network performance.
- **External Interference**: Particularly in wireless networks, external sources like other electronic devices or even neighboring Wi-Fi networks can lead to interference [31], reducing signal quality and overall network performance.
- **Malicious Attacks**: Denial of Service (DoS) attacks or other forms of cyberattacks can overwhelm network resources, causing severe degradation.

4.1.1.2 Manifestations of Service Degradation

How the aforementioned factors manifest in network traffic can vary, but common indicators include [5, 11]:

- **Increased Latency**: Often the first sign of degradation, latency refers to the delay in data transfer. Applications sensitive to delays, such as video calls or online games, are particularly affected.
- **Packet Loss**: Data travels in packets, and during degradation, some packets might not reach their destination. This loss can result in visible artifacts in video streams, glitches in audio calls, or even failed data transfers.
- **Reduced Throughput**: The speed at which data is transferred drops. This manifests as slower download or upload speeds and extended loading times for web pages.
- **Jitter**: Variability in latency can cause jitter. In real-time applications, like VoIP calls, jitter can cause voice disruptions or out-of-sequence packets, leading to poor call quality.
- **Connection Drops**: In extreme cases, devices might lose their network connection entirely or face frequent disconnections.

4.1.1.3 Impacts of Service Degradation

The ramifications of service degradation are multifaceted. For end-users, this means a diminished experience — videos buffer, calls drop, and websites load at glacial paces [18]. For service providers, degradation can lead to customer dissatisfaction, churn, and even financial losses in extreme cases. However, it is important to note, that different application types may have different thresholds for SD. While a flow that carries data for a live service may experience SD as soon as network conditions deteriorate slightly, other application types, like emails could be completely or mostly immune to deteriorated conditions. This underscores the necessity for an application-aware approach for SD identification.

4.1.2 The Latency-Service Degradation Nexus

4.1.2.1 Latency

In today's digital landscape, where online services from entertainment streaming to business conferencing are pervasive, understanding network performance is crucial. Latency stands as a primary metric for assessing this performance and is defined mathematically as:

$$\text{Latency (L)} = \text{Time (T2)} - \text{Time (T1)}. \tag{4.1}$$

Here:

- T1 represents the time when an action or request is initiated.
- T2 is the time when a response is received.

Latency significantly affects user experience quality, with even slight increases potentially leading to noticeable SD. I represent SD (ΔS) as a function of increased latency (ΔL):

$$\Delta S = f(\Delta L). \tag{4.2}$$

Here, f denotes the function that quantifies how service quality diminishes as latency increases. This present research aims to precisely identify this function while exploring other contributing variables.

4.1.2.2 Jitter

Besides latency, it is also beneficial to consider latency variability, or jitter, which is defined as the difference in latency measurements over successive intervals:

$$\text{Jitter (J)} = \text{Latency (L2)} - \text{Latency (L1)}, \tag{4.3}$$

where:

- L1 is the latency measured at an earlier time point.
- L2 is the latency measured at a subsequent time point.

The occurrence of high latency and jitter together may indicate more severe instances of SD. Therefore, examining both metrics is essential in my efforts to identify SD events.

4.1.2.3 Packet Inter-Arrival Time

PIAT quantifies the time interval between the arrivals of two consecutive packets within a network flow at the flow meter. For a sequence of packet arrival times $t_1, t_2, \ldots, t_n$, PIAT for the i-th packet is mathematically defined as:

$$\text{PIAT}_i = t_i - t_{i-1}$$

for $i > 1$, with $\text{PIAT}_1 = 0$ indicating no preceding packet for the first in the sequence.

PIAT is crucial in network analysis, serving various purposes from traffic characterization to behavior analysis. Its utility stems from providing detailed insights into traffic patterns and assisting in the detection of anomalies or disruptions within network flows. In this study, PIAT is instrumental due to its granularity and effectiveness in environments where monitoring resources may be limited. By leveraging PIAT, I aim to enhance the understanding of network dynamics and improve the precision of SD detection, making it an essential tool in my analysis methodology.

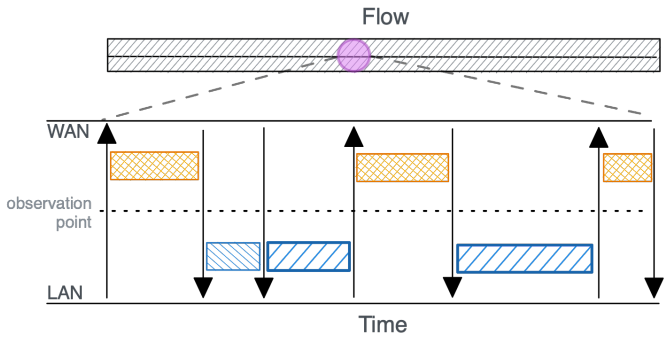


Figure 4.1: Visual representation of network traffic flow showing the vertical separation between WAN and LAN delays.

4.1.3 Vertical Separation of Delays

From the vantage point of a network edge, which serves as the observation point in this study, network flows network flow packets can be distinctly categorized into two primary directions: towards the LAN and towards the WAN. In the context of latencies determined via PIATs, a *vertical separation* of delays is essential for effective traffic monitoring and precise analysis of key metrics.

Isolating LAN-side delays is particularly important because external factors on the WAN side, which are beyond the network provider's control, can significantly influence the analysis results. Understanding local network conditions accurately necessitates focusing on LAN-specific metrics, as WAN-side delays introduce variability that can obscure the true performance of the residential network.

Figure 4.1 illustrates this concept clearly. The diagram shows the division of the bidirectional packet transfer within a flow, with PIATs measured in the WAN highlighted in orange with a crosshatch pattern to represent delays and interactions that occur outside the immediate control of the local network. Conversely, LAN PIAT components are depicted in blue, showcasing the internal network dynamics. Arrows in the figure indicate the direction of packet traffic, moving between LAN and WAN endpoints, providing a visual representation of data traversal across these network segments.

4.1.4 Leveraging PIAT for Latency Estimation

In my methodology, I utilize PIAT, measured in milliseconds, as a key metric for discerning network latency. PIAT values, along with packet direction and size measured in bytes, are captured as part of the Sub-Packet-Length-Time (SPLT) features measured by NFStream. These three features provide packet-level insights for the first n packets of each flow. For my study, I have set the recording of SPLT features to the maximum supported value of 255.

Traffic often appears as a burst of multiple packets; therefore, my analysis focuses on the time interval between the arrival of the last packet in such a burst (in the incoming direction) and the first outgoing packet that responds to this burst within the same flow. In Figure 4.1, instances meeting these conditions are highlighted with a filling pattern diagonal from the bottom left to the top right and with a thicker border.

Algorithm 1 Identifying LAN Delay

```
procedure ID_LAN_DELAY(dir, prev_dir, reversed)
    if reversed then
        ▷ dir: dst2src (LAN2WAN),
          prev_dir: src2dst (WAN2LAN) ◁
        return dir == 0 and prev_dir == 1
    ▷ dir: src2dst (LAN2WAN),
      prev_dir: dst2src (WAN2LAN) ◁
    return dir == 1 and prev_dir == 0
```

Algorithm 2 Vertical Separation of Flows

```
procedure VERT_SEP(splt_dir, splt_piat)
    $D \leftarrow []$
    prev_dir ← −1
    for idx, dir in enumerate(splt_dir) do
        if idx == 0 then
            prev_dir ← dir
        else
            ▷ Examining the dir and prev_dir I only take into account the response to the last packet in the
              case of a burst ◁
            if ID_LAN_DELAY(dir, prev_dir, reversed) then
                $D \leftarrow$ D + splt_piat[idx]
            prev_dir ← dir
    return $D$
```

I operate under the assumption that delays caused by local endpoints are negligible compared to those induced by broader network conditions. By selectively analyzing TCP flows, I ensure the PIAT values used reflect the time difference between the receipt of the last packet from the WAN side and the transmission of the corresponding TCP acknowledgment. This duration is indicative of the time traffic spends within the local network, effectively representing LAN-side latency. This latency is measured by analyzing the packet inter-arrival times of closely succeeding outgoing and incoming packets.

Algorithms 1 and 2 illustrate the steps involved in extracting PIAT values that pertain to LAN-side delays. While a similar analysis could also be conducted for LAN-to-WAN traffic, my focus remains on the LAN side due to its potential to provide detailed insights with fewer privacy concerns within a controlled monitoring environment. Understanding these internal dynamics lays a foundation for future methodologies that could extend to monitoring WAN-side behaviors.

The identification of LAN-side delays significantly narrows down the set of PIAT values compared to the original dataset. I discard flows with fewer than two LAN-side delays from further analysis because they either lack sufficient LAN-side delays to calculate jitter or cannot define a SD event. This filtering step results in a dataset that only includes flows with at least two LAN-side delays, as shown in Table 3.1.

Upon analyzing the temporal characteristics of this filtered dataset, I observed a notable decrease in flow arrival rates—from an average of approximately 1000 flows per second to around 400 flows per second—along with a more consistent distribution over time. Although some spikes in arrival rate persist, the overall data rate has slightly decreased, maintaining similar patterns to those observed prior to filtering. The proportion of flows with very few packets also decreased significantly; initially, almost 40% of flows contained only one packet, whereas now, only 20% contain fewer than ten packets. Similarly, the flow size has become more consolidated, with only 20% of flows containing less than 1 KB of data, down from 60% before filtering. Flow duration patterns also show fewer short-lived flows, with less than 5% lasting under 10 milliseconds compared to over 40% previously.

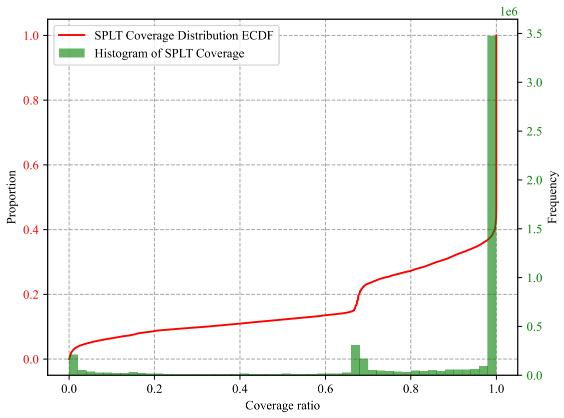


Figure 4.2: SPLT Coverage of the flow.

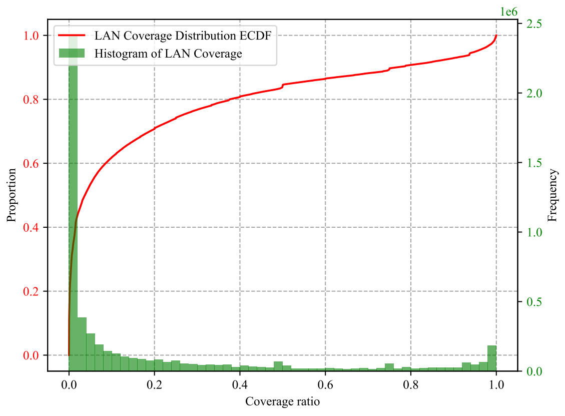


Figure 4.3: LAN delay coverage of the flow.

Despite these reductions, the overall pattern of the ECDF plots remains unchanged, suggesting that the main characteristics of the data were retained despite the narrower dataset scope.

4.1.5 Reduced Data View Coverage

A reduction in dataset size can undeniably impact the quality of data used for SD analysis. To validate the integrity of my filtered dataset and ensure that it remains representative of overall network behavior, I assess its coverage. High coverage would indicate that the majority of flows are captured within the initial scope of 255 packets of flows, an essential criterion given my reliance on SPLT values for PIAT-based latency analysis.

Figure 4.2 shows that a significant majority of the flows are fully encompassed within the initial SPLT values corresponding to the first 255 packets. Nevertheless, a few thousand longer flows exceed this packet limit, indicating partial coverage. Despite this, such partially covered flows constitute less than 5% of the total. Conversely, approximately 30% of flows are partially covered, while the vast majority either enjoy full coverage or are nearly fully covered.

Another metric to consider is the ratio of LAN delay durations to the remaining WAN delays in the measured flows, as illustrated in Figure 4.3. The analysis reveals that LAN delays are generally less prevalent than WAN delays, with some flows exhibiting significantly higher proportions of LAN delays. For 40% of the flows, LAN delays contribute almost nothing to their overall duration, indicating minimal LAN-side delays.

This analysis reassures us that, despite the dataset reduction, the primary characteristics and a substantial portion of the network dynamics are retained and adequately represented within the adjusted scope of this study.

4.2 Methodology for Identifying Service Degradation

In this section, I investigate the occurrence of SD events within the dataset, initially focusing on data from the most populous *Location 2* and limiting my analysis to the first three days to serve as a quasi-training set. This phase helps establish thresholds for

Table 4.1: Count of LAN delays per Application Category for *Location 2*

Application Category Name	**LAN Delay Count**
Web	9,016,470
Social Network	1,797,216
Cloud	1,177,916
Collaborative	743,453
Download	440,421
Network	264,156
Count of all delay samples	13,439,632

classifying SD events based on Interquartile Range (IQR) and Z-score analyses, which are detailed later.

I define SD as a statistically significant deviation from typical flow latency behavior, characterized by notable increases in latency and jitter. The thresholds for identifying these deviations are tailored to the specific requirements of different application categories. For instance, even minimal increases in latency might constitute SD in delay-sensitive applications like voice calls or remote desktop interactions, where timing is critical. Conversely, for activities such as downloading, where latency sensitivity is lower, an increase in delay may not be as perceptible.

Additionally, I explore prolonged SD events, defined as extended periods where deviations in latency and jitter persist. An SD event in this context is identified as a contiguous series of delays, starting with an initial outlier in jitter followed by subsequent delays that also qualify as outliers. This approach seeks to capture sequences where delays not only spike unexpectedly but also remain elevated above the established anomaly threshold. I further examine scenarios where, despite significant and sustained increases in delays, the flow continues to exhibit high jitter, maintaining levels above the SD threshold.

To ensure robustness and mitigate risks such as overfitting or unreliable threshold estimations, my analysis prioritizes application categories with sufficient data volume. Specifically, I focus on categories that have recorded at least 50,000 flows at *Location 2*. This criterion coincidentally aligns with the top six application categories in the entire dataset, which include: *Web*, *Social Network*, *Download*, *Cloud*, *Network*, and *Collaborative*.

Table 4.1 presents the LAN delay counts following this selection step. Notably, *Web* traffic constitutes the majority of the data, overshadowing the other categories. However, the remaining categories still provide a substantial number of samples, adequate for conducting reliable statistical analysis.

4.2.1 Interquartile Range Analysis

To evaluate the distribution of delays across various application categories, I utilized boxplots, as depicted in Figure 4.4. To enhance visibility of differences across distributions, the y-axis is set to a logarithmic scale. The boxplots are ordered in descending sequence based on the volume of delay samples, and mean values are denoted by orange rectangles on each plot.

The analysis reveals distinct distribution patterns within the application categories, categorizing them into two groups based on delay characteristics. Categories such as *Download* and *Network* typically show delays not exceeding 10 milliseconds, while others like *Cloud* often exceed 100 milliseconds at their upper whisker, with delays extending into the tens of milliseconds range. Notably, *Cloud*, *Download*, and *Network* also display delays in the

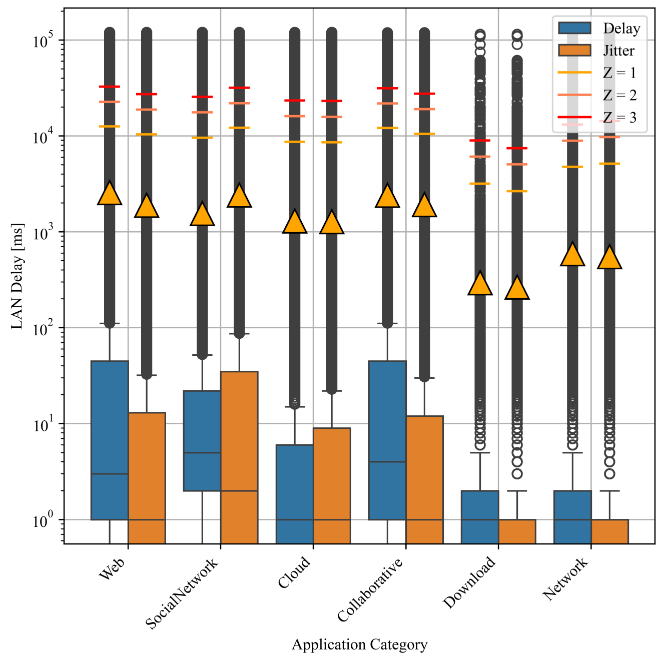


Figure 4.4: Distribution of LAN delay and jitter per application category.

sub-millisecond range, with medians set at 1 millisecond. Due to limitations in NFStream's measurement capabilities, delays under one millisecond are recorded as 0 milliseconds. The mean delays for *Download* and *Network* are notably below one second, which is significantly lower compared to other categories where means reach several seconds, largely due to the presence of outliers as determined by Interquartile Range Analysis (IQR) analysis.

Jitter distribution aligns closely with the observed delay patterns across all categories, with each showing a median jitter that matches or is lower than the median delay. The first quartile frequently registers at 0 milliseconds, indicating a significant occurrence of consistent delays (no jitter). Contrarily, *Social Network* flows display slightly longer jitters compared to delays, especially evident in the third quartile, upper whisker, and mean values.

Outliers, critical for identifying SD events, are evident across all categories and range from a few milliseconds to over 100 seconds. Detailed quantification of these outliers, including their impact on service quality, is presented in Table 4.2.

Table 4.2: Outlier Statistics for *Location 2*

Metric	**Quartile Analysis**	**Z-score Analysis**
Number of *delay* outliers	1,680,245	499,656
Number of *jitter* outliers	2,682,895	374,765
Number of *intersection* outliers	1,156,820	194,233
Total rate of *delay* outliers	12.502%	3.718%
Total rate of *jitter* outliers	19.963%	2.789%
Total rate of *intersection* outliers	8.608%	1.445%

Table 4.3: Mean and Standard Deviation Values for Each Application Category

Metric	Web	Social Network	Cloud	Collaborative	Download	Network
μ_{delay}	2,566.86	1,559.44	1,298.99	2,403.65	294.11	592.49
σ_{delay}	10,027.89	8,011.10	7,358.91	9,668.43	2,887.64	4,166.06
μ_{jitter}	1,893.55	2,424.25	1,277.34	1,921,77	265,79	551.51
σ_{jitter}	8,464.72	9,751.16	7,279.63	8,560.04	2,389.94	4,588.31

4.2.2 Z-Score Analysis

Next, I utilize Z-score analysis to detect outliers in my dataset, particularly focusing on unusually large delays indicative of SD. Positive Z-scores, which signify values above the mean, are of particular interest as they represent potential SD events. I adopt standard thresholds, considering Z-scores greater than 2 or 3 as significant, with the specific threshold dependent on the application category's sensitivity to delays.

Figure 4.4 presents delay and jitter means marked with orange diamonds and their corresponding Z-scores depicted by horizontal lines for all studied application categories at $Z = 1$, $Z = 2$, and $Z = 3$. These illustrate the degree to which specific measures deviate from the mean in standard deviation increments. The mean and standard deviation values crucial for these calculations are detailed in Table 4.3, highlighting that Z-scores typically mirror the distribution of mean values. For example, more sensitive categories such as *Web*, *Social Network*, *Cloud*, and *Collaborative* have Z-score thresholds with $Z = 3$ extending beyond 20 seconds. Conversely, the *Download* category's threshold is just under 10 seconds, while *Network* exceeds this slightly in both delay and jitter.

Figure 4.4 clearly demonstrates the stringent nature of Z-score analysis compared to IQR analysis. While IQR may flag a higher number of singular delay SD events (marked with black circles as outliers beyond the whiskers) due to its sensitivity to the lower limit, Z-score analysis bases its findings on deviations from the mean delay values, which are generally higher. Consequently, Z-score analysis results in fewer identified outliers. As shown in Table 4.2 for $Z = 3$, events exceeding this Z-score threshold are considered significant. In line with standard practices in anomaly detection, I adopt $Z = 3$ as the threshold for my Z-score analysis.

4.2.3 Examining Prolonged SD Events

SD events may manifest not only as singular delay and/or jitter outliers but also as prolonged sequences of consecutive outliers. To identify these sequences, I have developed an algorithm, outlined in Algorithm 3, that groups contiguous delays into a single SD event if all delays are recognized as outliers for a period defined by *Minimum Sequence Length* (`MIN_SEQ_LEN`), with an optional consideration for jitters. The algorithm operates as follows:

(i) It iterates through all delay samples in a flow.

(ii) If the jitter and delays are high (previously identified as singular SD points), it initiates an SD sequence.

Algorithm 3 Identifying LAN Delay

```
procedure FIND_SD_SEQUENCES(SDd_list, SDj_list, MIN_SEQ_LEN, require_jitter_for_sequence)
    for i, (SDd, SDj) in enumerate(zip(SDd_list, SDj_list)) do
        if require_jitter_for_sequence then
            seq_condition ← SDd  and SDj
        else
            seq_condition ← SDd
        if (sequence_length == 0 and SDj and SDd) or (sequence_length > 0 and seq_condition) then
            sequence_length ← sequence_length + 1
        else
            if sequence_length ≥ MIN_SEQ_LEN then
                start ← i - sequence_length
                end ← i
                sequences.append((start, end))
                seq_SD_list[start: end] ← [True] * (end - start)
            if sequence_length > 0 then:
                sequence_length ← 0
    ▷ If the last sequence goes until the end                                    ◁
    if sequence_length ≥ MIN_SEQ_LEN then
        start ← len(SDd_list) - sequence_length
        end ← len(SDd_list)
        sequences.append((start, end))
        seq_SD_list[start: end] ← [True] * (end - start)
    return [sequences, seq_SD_list]
```

(iii) The SD sequence continues as long as the delays (`SDd`) remain high, and optionally, if `require_jitter _for_sequence` is enabled, the jitter (`SDj`) must also remain high.

(iv) The sequence concludes when the traffic returns to normal, capturing the *start* and *end indexes* of the corresponding LAN delays marking the SD event.

Table 4.4: SD Event Count and Coverage Statistics for Different Minimum Sequence Lengths with `require_jitter_for_sequence` option turned *OFF*

`MIN_SEQ_LEN`	Web				Cloud				Download				Network				Collaborative				Social Network			
	Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*	
	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*
1	477,758	33	*126,702*	*24*	120,813	46	*21,033*	*28*	27,830	21	*7,304*	*16*	16,658	49	*1,705*	*19*	36,110	39	*17,491*	*32*	146,444	34	*19,599*	*18*
2	128,183	25	*54,624*	*18*	29,485	34	*6,281*	*14*	5,113	10	*116*	*5*	5,523	43	*240*	*8*	12,905	27	*5,170*	*20*	24,339	12	*1,933*	*5*
3	77,573	22	*42,125*	*17*	12,630	23	*745*	*4*	1,724	7	*63*	*4*	2,848	35	*174*	*7*	6,901	20	*2,318*	*15*	11,224	9	*1,330*	*5*
4	60,582	21	*37,374*	*16*	6,302	17	*541*	*3*	870	6	*45*	*4*	1,978	31	*125*	*6*	5,351	18	*2050*	*14*	5,664	7	*1084*	*4*
5	47,025	18	*29,675*	*14*	3,723	14	*309*	*2*	593	5	*41*	*4*	1,505	29	*94*	*4*	4,422	17	*1,874*	*13*	3,604	6	*963*	*4*
6	-	-	*13,404*	*9*	-	-	*156*	*2*	-	-	*27*	*3*	-	-	*43*	*2*	-	-	*1,667*	*12*	-	-	*730*	*4*
7	-	-	*10,924*	*8*	-	-	*128*	*2*	-	-	*27*	*3*	-	-	*41*	*2*	-	-	*1,576*	*12*	-	-	*561*	*3*
8	-	-	*6,665*	*6*	-	-	*116*	*2*	-	-	*26*	*3*	-	-	*23*	*1*	-	-	*219*	*2*	-	-	*538*	*3*
9	-	-	*5,704*	*5*	-	-	*113*	*2*	-	-	*22*	*3*	-	-	*17*	*1*	-	-	*141*	*2*	-	-	*514*	*3*
10	10,465	8	*5,290*	*5*	1,097	10	*109*	*1*	255	4	*22*	*3*	686	19	*17*	*1*	805	5	*131*	*2*	763	3	*485*	*3*
15	3,881	5	-	-	614	8	-	-	146	3	-	-	156	6	-	-	229	3	-	-	63	0	-	-
20	2,033	3	-	-	468	7	-	-	99	3	-	-	70	3	-	-	153	2	-	-	20	0	-	-
25	1,603	3	-	-	402	6	-	-	70	2	-	-	46	1	-	-	117	2	-	-	10	0	-	-

Figure 4.5 illustrates an example of an SD event for an application category requiring a minimum sequence length (`MIN_SEQ_LEN`) of 2. While an earlier LAN delay may also be an outlier, the absence of sufficient jitter and a subsequent outlier disqualifies it as an independent SD event.

This methodology allows us to assess SD events across all application categories, utilizing outliers identified by both IQR and Z-score analyses. I explore variations with the `require_jitter_for_sequence` option both enabled and disabled. Tables 4.4 and 4.5 present the counts of SD events and the percentage of total SD event time relative to the entire duration of the flow for each category. Only sequences meeting or exceeding

Table 4.5: SD Event Count and Coverage Statistics for Different Minimum Sequence Lengths with `require_jitter_for_sequence` option turned *ON*

`MIN_SEQ_LEN`	Web				Cloud				Download				Network				Collaborative				Social Network			
	Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*		Quartile		*Z-Score*	
	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*	count	cov.%	*count*	*cov.%*
1	506,144	17	*126,757*	*9*	126,392	27	*21,034*	*20*	28,619	20	*7,312*	*12*	18,664	40	*1,709*	*13*	37,910	22	*17,495*	*17*	149,818	28	*19,607*	*13*
2	74,708	6	*139*	*0*	19,636	11	*1*	*0*	4,663	9	*10*	*0*	5,956	32	*28*	*0*	7,922	5	*3*	*0*	19,629	5	*99*	*0*
3	26,105	3	*33*	*0*	6,259	5	*0*	*0*	1,300	6	*1*	*0*	2,786	22	*0*	*0*	3,925	1	*1*	*0*	6,423	2	*3*	*0*
4	15,323	3	*1*	*0*	2,652	2	*0*	*0*	664	4	*0*	*0*	1,608	16	*0*	*0*	2,841	1	*0*	*0*	2,326	1	*0*	*0*
5	10,053	2	*0*	*0*	1,351	1	*0*	*0*	414	4	*0*	*0*	1,019	12	*0*	*0*	2,072	1	*0*	*0*	1,178	1	*0*	*0*
10	2,437	2	-	-	138	0	-	-	159	4	-	-	189	4	-	-	325	0	-	-	64	0	-	-
15	1,211	2	-	-	29	0	-	-	89	2	-	-	40	2	-	-	23	0	-	-	5	0	-	-
20	865	2	-	-	13	0	-	-	64	2	-	-	13	0	-	-	4	0	-	-	1	0	-	-
25	774	1	-	-	7	0	-	-	33	1	-	-	6	0	-	-	0	0	-	-	0	0	-	-

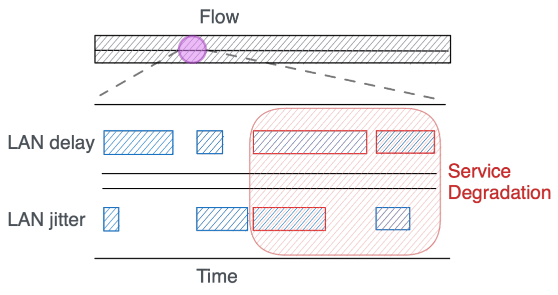


Figure 4.5: A sample SD event when `MIN_SEQ_LEN` = 2 and `require_jitter_for_sequence = False`

the stipulated minimum length are considered in the final count of SD events. Longer sequences, even though they extend beyond this threshold, are counted as a single event.

I observe a decreasing trend in the number of SD events and their duration coverage as the minimum sequence length (`MIN_SEQ_LEN`) requirement is increased. Notably, with Z-score analysis, the initial count and coverage of SD events are significantly lower compared to those identified through IQR analysis. A similar pattern of counts and coverages is reached by `MIN_SEQ_LEN` = 3 for most categories, except for the *Web* category, which shows a consistent pattern with IQR analysis, indicating more prolonged and extreme SD events in this category that are detectable by both methods.

The reduction in SD events is neither linear nor uniform across different application categories and the two outlier identification methods. IQR analysis generally shows a steadier decrease in SD events, whereas Z-score analysis often experiences abrupt declines in SD event counts and coverage, which then stabilizes. This behavior varies by application category at different `MIN_SEQ_LEN` values—*Download*, *Network*, and *Social Network* at 2; *Cloud* at 3; *Collaborative* at 8. Interestingly, the *Web* category does not exhibit such a drastic drop, suggesting a characteristic resilience in the length of its SD events.

Additionally, when examining SD events identified by delay-jitter outliers compared to those identified by delay outliers alone, I observe similar patterns. At `MIN_SEQ_LEN` = 1, the number of SD events is slightly higher, attributed to shorter but more frequent sequences disrupted by periods of low jitter, leading to a decrease in overall SD coverage. As the required sequence length increases, the count of SD events sharply declines, necessitating longer sequences for classification as an SD event. With IQR analysis, by

Table 4.6: Minimum Sequence Lengths for SD Event Identification per Application Category

Application Category	`MIN_SEQ_LEN`
Web	6
Cloud	3
Download	2
Network	2
Collaborative	8
Social Network	2

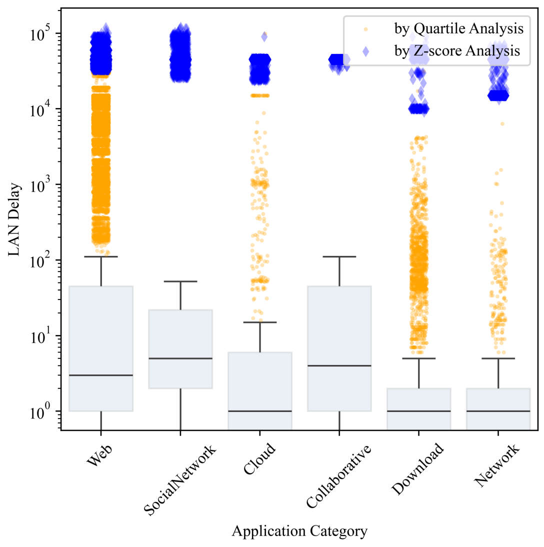


Figure 4.6: Distribution of the delay composition of SD events.

`MIN_SEQ_LEN` = 10, I observe coverage levels previously seen only at lengths of 25, with most categories falling below 5% except for *Network*, which maintains over 10% coverage. Conversely, for Z-score analysis incorporating jitter requirements, no SD sequences are identified in any category beyond `MIN_SEQ_LEN` = 5. Indeed, by `MIN_SEQ_LEN` = 4, only one SD event remains, and even at lengths of 2 and 3, only a handful of SD events are present. This suggests that a dual requirement for high delay and high jitter may be overly restrictive for identifying SD events in this context.

I have made the code for the detailed analysis discussed in this section, including a comprehensive set of plots, available as digital artifacts [13].

4.2.4 Outlier Thresholds for Reliable SD Detection

For setting the outlier identification threshold, I opt for the Z-Score method, which provides a more conservative estimate, resulting in lower false positive rates. To determine the Minimum Sequence Length for SD events, I adopt an empirical approach, selecting a unique sequence length for each application category that reduces the SD coverage rate to below 10%. These thresholds are highlighted in Table 4.4 in red, indicating the first instance where coverage percentages drop below this critical threshold. The specific values for each category are summarized in Table 4.6, aiming to further minimize false positive SD events.

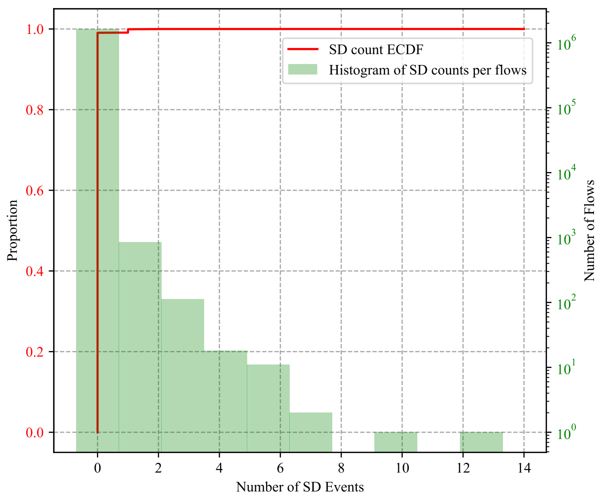


Figure 4.7: Distribution of number of SD events across flows.

These sequence lengths correlate with the noticeable declines in SD coverage discussed earlier in Section 4.2.3. The selected LAN delays for these thresholds are depicted in Figure 4.6, which contrasts the LAN delay points identified in each application category by the most stringent IQR analysis (requiring a minimum of 25 consecutive high LAN delay and jitter) against those identified by the chosen Z-score method. This comparison illustrates that while IQR analysis tends to identify many comparatively minor delays as part of prolonged SD events, Z-Score analysis selectively identifies more extreme values by default, emphasizing its stricter criteria.

4.2.5 Effectiveness of Z-Score Analysis in Identifying SD Events

In this section, I analyze various statistics of the SD events identified using the chosen Z-score method. Figure 4.7 presents the distribution of SD event counts across flows. The y-axis, represented on a logarithmic scale, indicates the number of flows, while the x-axis shows the number of SD events, ranging from 0 to over 14. The majority of flows either have no SD events or only a few, with an ECDF plot revealing that approximately 99% of flows contain at most one SD event, making multiple events within a single flow exceedingly rare.

Table 4.7 details the number of flows, number of SD events, and the number of flows with at least one SD event across all application categories, alongside the proportion of flows with SD events relative to the total number of flows. Consistent with Figure 4.7, only a small fraction of flows contain multiple SD events, as indicated by the slight difference between *Total SD Events* and *Flows with SD*. The proportion of flows experiencing SD remains below 1% for all categories, marginally exceeding this threshold only in the *Web* category, which accounts for the majority of identified SD events.

Figure 4.8 illustrates the temporal distribution of SD events, with the x-axis displaying time in milliseconds and the y-axis depicting the number of SD events at each time point. This plot aims to determine whether SD events are evenly distributed over time or if certain periods contain higher concentrations of events. Notable spikes and valleys in the data suggest moments where SD event counts were significantly higher or lower, with Monday showing frequent fluctuations, but the most intense activity observed on Wednesday just before the measurement period concluded. This could indicate a specific time

Table 4.7: Distribution of SD Events Across Application Categories

	Total SD Events	Flows with SD	Proportion with SD
Web	13,404	12,450	1.2434%
Social Network	1,933	1,746	0.8620%
Cloud	745	653	0.3844%
Collaborative	219	215	0.2925%
Download	240	208	0.2715%
Network	116	116	0.1062%

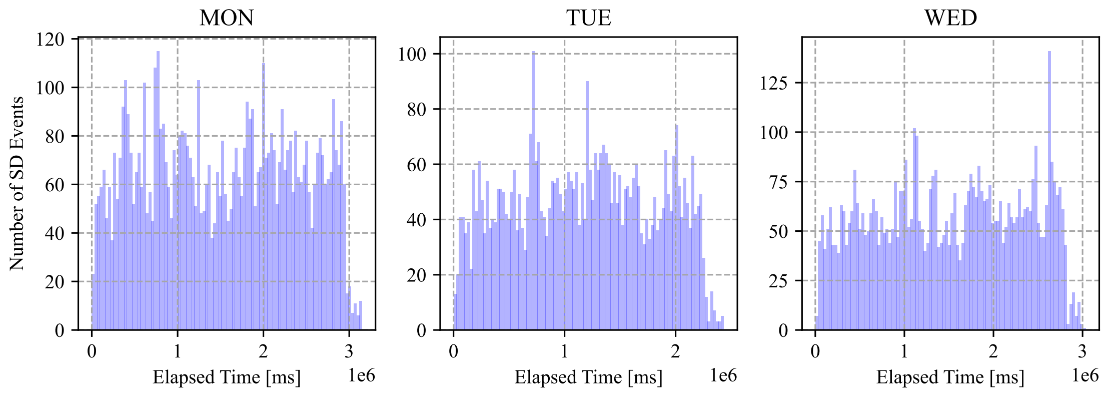


Figure 4.8: Distribution of number of SD events across flows.

when network disturbances were more pronounced, affecting multiple flows. Additionally, a common trend on all days is a marked decrease in SD events towards the end of the measurement period, possibly reflecting characteristics of the measurement timeframe or the impact of the measurement process itself, such as truncating ongoing flows.

4.3 Generalizability

With the insights gained from my analysis at *Location 2*, in this section, I extend the Z-score analysis for SD identification to *Location 1* and *Location 3*. This extension aims to examine the consistency of delay characteristics and SD events across different locations, thereby assessing the generalizability of my approach. For each location, I calculate Z-scores for delay and jitter across all samples, identifying outliers where $Z > 3$. Following the established methodology from *Location 2*, I search for sequences of SD events initiated by outliers in delay and jitter, where high delay behavior persists. I then determine the smallest `MIN_SEQ_LEN` that reduces the coverage of these events below 10% for each category, comparing these values with those derived from *Location 2*. The findings and their implications for generalizability are detailed in Section 4.3.1.

To validate my approach, I apply the same analytical steps to the remaining holdout portion of the dataset, which includes flow data from the last two days of the experiment (Thursday and Friday). These days were chosen because they exhibited similar flow-arrival rates and data rates as observed previously (see Section 3.2.2). By analyzing the outcomes, I aim to confirm the consistency of the observed patterns and validate the applicability of my defined methodology during these periods. The results of this validation process are explored in Section 4.3.2.

4.3.1 Cross-Location Validation of Service Degradation

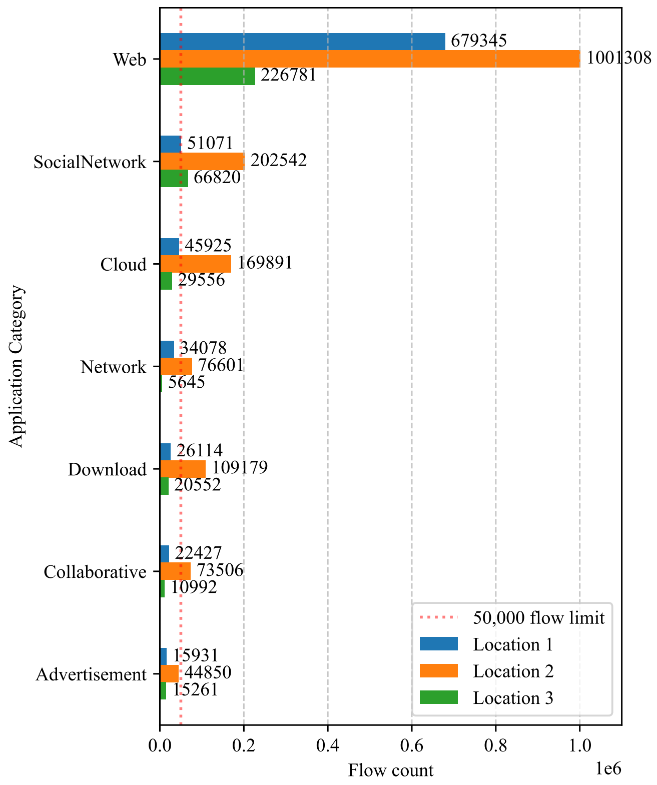
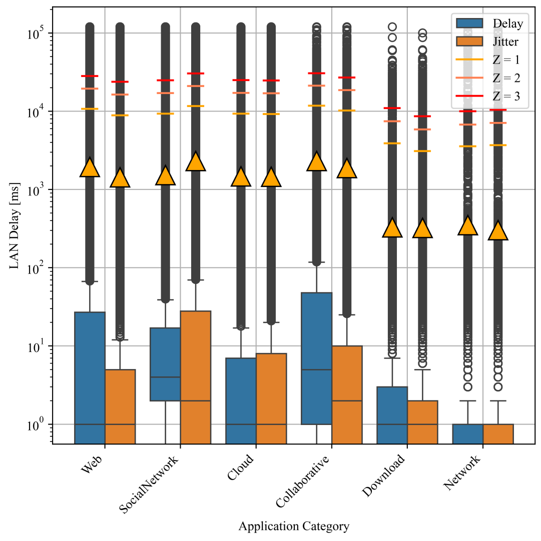


Figure 4.9: Application category cardinality for all locations.

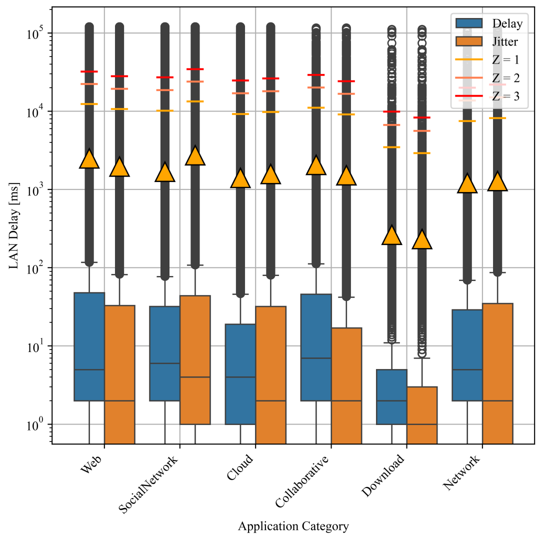
Table 4.8: Count of LAN delays per Application Category for All Locations

Application Category Name	**Location 1**	**Location 2**	**Location 3**
Web	3,453,780	9,016,470	2,017,416
Social Network	490,886	1,797,216	555,845
Cloud	330,038	1,177,916	197,503
Collaborative	256,976	743,453	109,425
Download	143,806	440,421	110,501
Network	109,343	264,156	38,057
Count of all delay samples	4,784,829	13,439,632	3,028,747

To ensure consistency in my analysis, I continue to focus on the same application categories as those analyzed in data from *Location 2*. Figure 4.9 compares the flow counts across the three locations, from which I find that only the *Web* and *Social Network* categories at *Location 1* and *Location 3* meet the 50,000 flow count threshold set in my original analysis. Lower flow counts in the other categories may influence SD event characteristics, skewing them toward the behavior observed in fewer flow records, which may not be as representative as those derived from higher flow counts. Despite these limitations, I proceed with the analysis for all previously chosen categories— *Web*, *Social Network*, *Cloud*, *Network*, *Download*, and *Collaborative*—keeping in mind that the results for categories with lower sample counts may be less reliable. By examining the differences in SD event requirements between categories with varying flow counts, I aim to assess whether the 50,000 flow threshold is justifiable or could potentially be lowered.



(a) Location 1.



(b) Location 3.

Figure 4.10: Distribution of LAN delay and jitter for *Location 1* and *Location 3*.

Table 4.9: Mean and Standard Deviation Values for Each Application Category

(a) Location 1

Metric	**Web**	**Social Network**	**Cloud**	**Collaborative**	**Download**	**Network**
μ_{delay}	1,945.08	1,527.17	1,475.12	2,314.52	329.43	351.44
σ_{delay}	8,761.14	9,421.93	7,836.64	9,421.93	3,551.12	3,205.05
μ_{jitter}	1,432.03	2,314.18	1,454.34	1,885.77	323,33	303.34
σ_{jitter}	7,430.50	9,335.86	7,767.23	8,353.78	2,764.91	3,382.51

(b) Location 3

Metric	**Web**	**Social Network**	**Cloud**	**Collaborative**	**Download**	**Network**
μ_{delay}	2,495.82	1,684.57	1,212.67	2,070.17	264.63	1,212.67
σ_{delay}	9,888.79	8,474.79	7,777.25	9,001.14	3,203.27	6,253.60
μ_{jitter}	1,969.48	2,739.77	1,580.08	1,520.29	233,12	1,286.55
σ_{jitter}	8,673.23	10,574.94	8,197.80	7,572.17	2,683.39	6,879.52

The LAN delay counts across all application categories (see Table 4.8) exhibit similar patterns to the flow count statistics, albeit falling short of the delay counts observed at *Location 2*. Notably, *Location 3* shows significantly lower counts compared to *Location 1*, except for the *Social Network* category.

Figure 4.10 and Table 4.9 depict the delay and jitter distributions for *Location 1* and *Location 3*, revealing patterns virtually identical to those at *Location 2*. The distributions across application categories maintain similar relative delays, with only minor differences: at *Location 1*, *Web* delays occasionally dip below 1 millisecond, and at *Location 3*, the 75th-percentile of *Network* delays stay below 1 ms, with only outliers exceeding this threshold. These observations suggest that while lower flow counts might slightly bias the results towards lower delays, the fundamental characteristics of delay distribution are preserved, even with significantly reduced data volumes.

Interestingly, *Location 3* experiences consistently higher delay values across all categories, with the 25th-percentile values increasing by several milliseconds, leading to a broader distribution in jitter. This pattern could reflect the unique network traffic characteristics of *Location 3*, possibly due to its more remote location compared to the other sites, which are within the same municipality. Despite these variances, the higher percentiles, mean values, and Z-scores show comparable characteristics, affirming the overall pattern consistency.

To delve deeper into longer SD events characterized by consecutive high delays, I replicate the specific scenario analyzed at *Location 2*. In this scenario, an SD event begins with an outlying delay and jitter (Z-score > 3), with the high delay persisting throughout the event. I progressively increase the minimum sequence length required to qualify as an SD event, analyzing both the number and coverage percentage of these events. The results of this extended analysis are presented in Table 4.10.

Although the total counts of SD events decrease significantly when stricter criteria are applied, the coverage percentages exhibit a decay pattern very similar to that observed at *Location 2*, differing by only a few percentage points. Furthermore, when examining key thresholds—specifically, points where coverage falls below 10% and where significant drops in coverage occur—I identify the same critical sequence lengths as those previously established in Table 4.6.

This consistency across different locations not only confirms that extreme traffic patterns leading to SD events are comparable, if not identical, across the three locations, but it also validates the robustness of my analytical approach. Moreover, it suggests that this methodology can effectively be applied to application categories with significantly lower flow counts, reinforcing the reliability and appropriateness of my chosen methods for analyzing SD events.

4.3.2 Service Degradation Validation via Holdout Set

For my SD event analyses thus far, I have utilized data from the first three measurement days designated as training data. To extend my validation, I leverage the data from the last two days as testing data in a model training-testing scenario. This test data allows us to validate the SD event identification approach I applied by comparing results across different days, similar to my analyses across various locations. By tallying the results and quantifying differences in thresholds and `MIN_SEQ_LEN`, I aim to confirm the robustness and consistency of the methodology I used.

I have chosen not to include delay and jitter distribution plots for the test data from the three locations in this document due to their high similarity to the training data distributions. However, to ensure transparency and facilitate further research, I provide the code for my detailed analysis, including a comprehensive set of plots, as digital artifacts [13]. Upon examining the delay and jitter distributions in the test data, I observe highly sim-

Table 4.10: SD Event Count and Coverage Statistics for Different Minimum Sequence Lengths with `require_jitter_for_sequence` option turned *OFF* using the Z-Score Identification Method

(a) Location 1

	Web		**Cloud**		**Download**		**Network**		**Collaborative**		**Social Network**	
`MIN_SEQ_LEN`	count	cov.%	count	cov.%	count	cov.%	count	cov.%	count	cov.%	count	cov.%
1	43,057	24	6,602	27	856	15	405	16	6,017	32	5,272	18
2	16,366	18	1,901	13	35	8	35	6	1,761	19	611	6
3	12,642	17	218	4	24	7	30	6	671	13	467	6
4	10,893	16	176	3	19	7	28	6	614	13	384	5
5	8,619	14	115	3	18	7	25	5	571	12	344	5
6	4,213	9	77	2	17	7	10	3	517	11	220	4
7	3,274	8	58	2	17	7	9	3	474	11	143	3
8	1,974	6	44	2	17	7	4	1	66	2	131	3
9	1,707	5	43	2	17	7	3	1	50	2	124	3
10	1,577	5	41	2	17	7	3	1	50	2	109	2

(b) Location 3

	Web		**Cloud**		**Download**		**Network**		**Collaborative**		**Social Network**	
`MIN_SEQ_LEN`	count	cov.%	count	cov.%	count	cov.%	count	cov.%	count	cov.%	count	cov.%
1	27,154	24	3,774	25	825	14	529	21	1,960	32	6,293	19
2	10,213	17	894	11	30	7	68	9	695	23	501	5
3	8,005	16	113	3	15	6	60	9	382	19	376	4
4	7,072	15	90	3	11	5	51	8	329	18	313	4
5	5,667	13	62	3	10	5	37	6	309	17	266	4
6	2,804	9	41	2	9	5	7	2	278	16	201	3
7	2,202	8	38	2	9	5	6	2	246	15	132	2
8	1,374	6	36	2	8	5	5	1	58	5	124	2
9	1,164	5	35	2	8	5	1	0	46	5	120	2
10	1,037	5	33	2	8	5	1	0	43	5	117	2

ilar patterns to those in the training data, albeit with slight variations in median values. Notable changes between the training and testing datasets for all locations include:

- *Location 1* displays slightly higher *Web* delays paired with lower jitter, an increase in lower-end *Social Network* delays, and lower delays in the *Cloud*, *Download*, and *Network* categories, along with correspondingly lower jitter distributions.
- *Location 2* shows virtually identical distributions with a broader spread in jitter within the *Network* category.
- *Location 3* retains the higher delay and jitter distribution observed during the training phase due to its geographical distance, yet maintains highly similar patterns, with the notable exception of increased *Download* and *Network* delays and jitter.

These observations suggest that my methodology yields consistent and reliable results across different testing conditions, further validating the soundness of my chosen analytical approach.

Table 4.11 illustrates the relative differences (expressed as percentages) between the measured mean and standard deviation across all locations, upon which the Z-Score method bases its outlier detection. These differences are color-coded according to their magnitude, with larger deviations highlighted in darker shades of red.

Examining the mean and standard deviation statistics for delay and jitter, I observe the most significant deviation in the delay mean: a 45% decrease in the *Network* category at *Location 2*. The most substantial change in jitter mean occurred in the *Collaborative* category at *Location 1*, where the average jitter was over 75% higher compared to the training data. The greatest changes in standard deviation were 31% for delay and 42% for jitter, both observed in the *Collaborative* category at *Location 1* and *Location 3*, respectively. The *Collaborative* category generally exhibited the most variability in delay and jitter changes between the training and test data, with other notable variations in the *Social Network* and *Cloud* categories. Changes in other categories were slight or negligible.

Table 4.11: Mean and Standard Deviation for Each Application Category in the Test Data given as a Relative Percentage Difference to Training Data

	Metric	**Web**	**Social Network**	**Cloud**	**Collaborative**	**Download**	**Network**
Location 1	μ_{delay} (Δ%)	20.92	-9.67	-19.15	33.00	-4.63	-4.49
	σ_{delay} (Δ%)	9.41	-3.07	-4.55	30.94	-1.29	-0.08
	μ_{jitter} (Δ%)	24.28	-7.50	-27.68	78.74	-7.22	-1.49
	σ_{jitter} (Δ%)	10.51	-2.09	-7.17	52.98	-3.57	1.91
Location 2	μ_{delay} (Δ%)	7.14	44.51	14.31	-45.05	14.06	-9.12
	σ_{delay} (Δ%)	2.60	19.06	-6.62	-28.88	7.06	-6.89
	μ_{jitter} (Δ%)	-3.09	26.26	-22.06	-48.93	36.04	-8.18
	σ_{jitter} (Δ%)	-3.16	10.53	-25.57	-29.44	16.53	-6.96
Location 3	μ_{delay} (Δ%)	1.62	-29.46	-8.16	20.60	-1.69	3.61
	σ_{delay} (Δ%)	1.16	-16.57	-16.21	27.58	-1.43	2.80
	μ_{jitter} (Δ%)	2.87	-30.45	53.57	43.31	-7.71	-3.24
	σ_{jitter} (Δ%)	2.39	-17.53	15.90	41.87	-4.16	-0.14

These differences were fairly evenly distributed across the three locations, with *Location 1* experiencing the most significant changes. When testing for the optimal minimum sequence length using the same parameters, the exact same thresholds as those determined from the training data were obtained. Table 4.12 details the coverage percentages for all three locations and all six application categories analyzed. Despite slight variations in the exact SD event coverage percentages, the `MIN_SEQ_LEN` for all categories was consistent across all locations, with the significant decrease in coverage dipping below 10% at matching minimum sequence lengths. The distribution of these results was identical to that observed in the training data (see Table 4.6), confirming the reliability and consistency of my methodology across different datasets.

4.4 Discussion

This section explores potential limitations and assesses the validity of my findings, with particular attention to their applicability across various domains and their implications for the Quality of Experience (QoE) perceived by users.

Table 4.12: SD Event Coverage Statistics (*cov.%*) for Different Minimum Sequence Lengths with `require_jitter_for_sequence` option turned *OFF* using the Z-Score method run on the test data (*L1*, *L2* and *L3* stand for *Locations 1*, *2* and *3* respectively)

MIN_SEQ_LEN	**Web**			**Cloud**			**Download**			**Network**			**Collaborative**			**Social Network**		
	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3
1	22	24	26	27	28	24	14	19	12	18	20	18	31	33	35	18	19	21
2	17	18	19	14	14	11	8	3	3	4	7	8	20	21	26	6	6	7
3	16	17	17	4	5	4	6	3	2	2	5	7	15	16	23	5	5	6
4	15	16	16	3	4	4	6	3	2	2	5	6	14	15	22	5	5	6
5	13	14	14	3	3	3	6	2	1	1	3	5	14	14	22	4	4	5
6	9	9	8	2	3	2	6	2	1	1	1	1	13	13	20	4	4	4
7	8	8	7	2	2	2	6	2	1	1	1	1	12	13	19	3	3	4
8	6	6	5	2	2	2	6	2	1	0	1	1	2	2	5	3	3	3
9	6	6	5	2	2	2	6	2	1	0	0	0	2	2	5	3	3	3
10	5	6	5	2	2	2	6	2	1	0	0	0	2	2	4	3	3	3

The research utilized LAN data from a university dormitory, which may exhibit network traffic characteristics distinct from other environments such as home networks or small-office/home-office (SOHO) settings. These environments often encounter resource constraints that could differently affect the detection and analysis of SD events. Conversely, analyzing SD in large enterprise networks could yield significant benefits in terms of financial and resource management. However, my findings have not been directly tested in these varied settings, and it is common for studies to encounter challenges when attempting to translate their models across different environments. To address this, my study employed a university dormitory setting for data collection, hypothesizing that it provides a hybrid representation of residential and institutional network traffic—merging casual internet usage by residents with university-related activities such as accessing cloud resources or remote sessions.

By leveraging data that potentially spans multiple domains, my aim was to enhance the robustness and applicability of my SD event identification method to both home and enterprise environments. Nevertheless, this assumption is speculative and requires further empirical validation. Specifically, in home environments, an experiment over a significantly longer time span would be necessary to gather a comparable volume of data. The choice of a dormitory setting was primarily driven by the convenience of collecting a large and diverse dataset.

An additional aspect of this research involved evaluating the generalizability of my findings by applying the methodology in different locations and at various times. These experiments validated my model by producing results with minimal deviation, as expected from the highly similar flow characteristics and behaviors observed across locations and over time. To rigorously confirm the robustness of the chosen method, testing on data sampled at significantly different times and from varied university dormitory environments might be necessary. Despite some variations, the data confirmed the model's validity within the current settings. My findings also demonstrated that the methodology is effective even with considerably lower flow counts, indicating potential applicability in smaller environments.

My empirical definition of an SD event is based on the occurrence of prolonged incidents characterized by statistically extreme delays and jitter. This approach aims to identify events that are most likely to significantly impact the perceived network service quality. By focusing on sustained extreme events, I sought to adopt a more conservative estimate to further limit false positives. However, whether these identified SD events translate to actual impacts on user QoE was not directly assessed in this study. Investigating this relationship would require a more controlled experiment or the integration of expert knowledge, both of which were outside the scope of this study and remain areas for future research. My method provides a conservative statistical framework that operates effectively in an unsupervised manner.

Chapter 5

Intra-Flow Latency-Induced Service Degradation Detection

This chapter explores the home network environment focusing on the limitations of residential network devices. Through a horizontal separation of the flows collected and completed with SD events using the statistical heuristic presented in Chapter 4 the network data is split into an O and NO part. I focus on following statistical analysis regarding different split metrics, I build and evaluate various models for a classification task (with the presence of SD in the NO part as target) and 4 regression problems (with the target of predicting the amount of SD events in the NO parts and the length, start and end indices of its longest SD event).

My work specifically focuses on flows measured at *Location 2*. This focus helps to reduce the dataset's size, simplifying our analysis and making the evaluation more straightforward. Nonetheless, as indicated by the results in Section 4.4, the findings are transferable to other measured locations and, by extension, to similar environments.

The rest of this chapter is organized as follows: Section 5.1 introduces the foundational concepts and preparatory considerations, including the methods and metrics used to identify and handle service degradation (SD) events through the O/NO split. Section 5.2 details my analysis, focusing on threshold analysis for effective LAN delay monitoring, optimal O/NO split thresholds, and the distribution of flow features following the O/NO split. Section 5.3 evaluates the predictive power of the observable parts of network flows, covering data preparation, experimental design, and model performance metrics for both classification and regression tasks, along with a feature importance analysis of the best classification model. Section 5.4 discusses the implications of my findings, focusing on predictive performance and optimal thresholds, LAN delays, variable delay thresholds, model performance limitations, and practical considerations, while also suggesting future research directions.

As with the case of the SD analysis, all source codes and the evaluation results of the classification and regression models are available as a supportive material at [14].

5.1 Foundational Concepts and Preparatory Considerations

5.1.1 Network Flow Monitoring in Constrained Environments

The notion of monitoring networks is not novel. It has long been acknowledged as an imperative for maintaining network health [34, 24, 29]. The procedure of network flow measurement is described in Section 3.1. However, monitoring becomes a Herculean task in constrained environments like home routers.

Several challenges punctuate this scenario:

- **Hardware Limitations**: Home routers are often not designed with extensive monitoring in mind. Their primary task is to route traffic efficiently. Monitoring every packet, especially in high-traffic scenarios, can overburden their limited computational capabilities.
- **Diverse Traffic Patterns**: With the proliferation of IoT devices, gaming consoles, streaming devices, and traditional computing hardware, home networks are a melting pot of diverse traffic patterns. Identifying anomalies or degradation patterns amid this diversity is challenging.
- **Real-time Analysis Needs**: Detecting and mitigating service degradation requires real-time or near-real-time analysis. Given the hardware constraints, achieving this real-time insight is often not feasible.
- **Storage Constraints**: Storing logs for deep analysis might seem like a solution. However, storage, like processing power, is a luxury in many home routers.

While solutions like offloading monitoring to cloud servers or using dedicated monitoring hardware exist, they come with privacy concerns or additional costs. Thus, the conundrum remains — how does one monitor effectively in environments inherently ill-equipped for the task?

5.1.2 Hardware Offloading in Network Devices

Network devices utilize a dual-path architecture to handle packet processing efficiently: the fast path, which employs dedicated hardware for forwarding tasks, and the slow path, which relies on CPU resources for more complex processing. Depicted in Figure 5.1, this design effectively caters to the distinct requirements of data plane and control plane tasks, where packets requiring minimal processing are directed through the fast path, and those necessitating more extensive processing are handled by the slow path [22, 7, 37].

Typically, the initial packets in a network flow are processed by the CPU. This initial slow-path processing involves critical tasks such as forwarding lookup to determine the egress port/zone, NAT policy lookup, and security policy enforcement. The CPU's processing capabilities enable thorough inspection and enforcement of policies. However, relying on the CPU for packet forwarding across all network flows can be resource-intensive and may lead to performance bottlenecks under heavy traffic loads.

To mitigate these performance issues, certain flows that meet specific criteria are offloaded to specialized hardware for fast-path processing. In connection-oriented protocols like TCP, flows might be flagged for fast-path processing after the connection is fully established, which typically occurs after the TCP three-way handshake. In contrast, for connectionless protocols like UDP, flows can be offloaded to the fast path after processing the initial packet, as there is no formal connection establishment process. Additionally, some flows might be flagged for fast-path processing once encryption tunnel establishment is complete.

Once the necessary initial packets are processed, subsequent packets in the same flow can be offloaded to specialized hardware for fast-path processing. This offloading significantly reduces the CPU's workload, enhancing the device's overall performance. The dedicated hardware processor then handles the remaining packets in the flow, ensuring swift and efficient processing.

However, certain flows may require specialized network functions that cannot be fully addressed by inspecting only the initial packets. These functions include complex analysis and inspection tasks such as application control, device intelligence, deep packet inspection (DPI), and intrusion prevention system (IPS) functions.

To maintain necessary visibility over network flows, network devices can be configured to handle flows in the slow path up to a certain threshold before offloading them to hardware-accelerated processing. This approach ensures that essential visibility and control for performing network functions are retained. While the CPU manages the traffic, detailed inspection and comprehensive monitoring are possible, providing deep visibility into the network.

The implementation of offloading mechanisms varies across different vendors and their product lines, each differing in technical, methodological, and functional details. For instance, some implementations might support flagging flows that meet specific criteria in the fast path to be redirected back to the CPU for more detailed inspection. Other implementations may utilize more than the initial packets of a flow before offloading to hardware-accelerated processing. Criteria for fast-path processing might include trusted applications or low-risk traffic, and latency-sensitive applications requiring minimal delay.

The dual fast-path slow-path approach underscores the inherent challenges of network monitoring in resource-constrained environments. By leveraging the strengths of both hardware and software, it facilitates the implementation of simple, time-critical algorithms in hardware and more complex network functions in software. Balancing hardware offloading and CPU processing enables network devices to achieve high performance while retaining essential control and visibility, tailored to the specific needs and behaviors of network traffic.

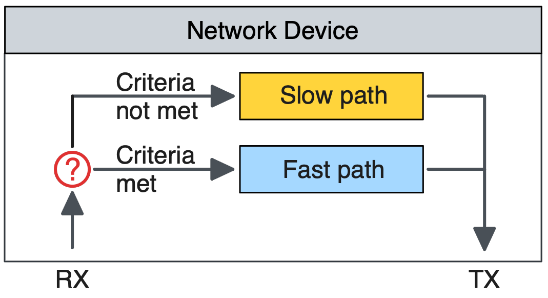


Figure 5.1: Generic architecture of a network device with hardware offloading capabilities.

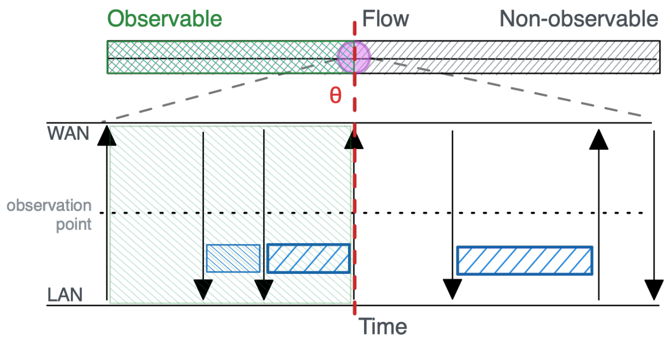


Figure 5.2: Illustration of the horizontal separation of a network flow into observable and non-observable segments.

5.1.3 Horizontal separation of Flows

In this study, the decision to keep a flow in the software layer or transition it to the hardware layer is governed by a threshold parameter, θ. This threshold defines the packet limit within which each flow should be monitored by the software layer. Once a flow's packet count surpasses this threshold, it transitions to the hardware layer.

Mathematically, if p represents the number of packets in a flow, then:

$$\text{State of Flow} = \begin{cases} \text{Observable} & \text{if } p \leq \theta \\ \text{Non-observable} & \text{if } p > \theta \end{cases}$$

This threshold-based differentiation can be understood as the horizontal separation of communication space. The θ parameter determines how many packets of a given flow are considered observable. Based on this parameter's value, two scenarios can occur:

(i) If the size of a given flow (number of packets) is less than or equal to the threshold, $||f_i|| \leq \theta$, where f_i denotes the i^{th} network flow, all packets of the flow are observed.

(ii) For $||f_i|| > \theta$, the observed packets are denoted by O_p and the unobserved by NO_p. Individual packets are represented as p_j, where j indicates the chronological order. The flow f_i then breaks down into two subsets: $O_p = \{p_1, \ldots, p_\theta\} \subseteq f_i$ and $NO_p = \{p_{\theta+1}, \ldots, p_{||f_i||}\} \subseteq f_i$.

Figure 5.2 illustrates a flow separated into observable and non-observable segments based on the θ parameter. In this figure, the observable segments are highlighted in green, indicating the portion of the flow that remains within the software layer. The non-observable segments are shown in gray, representing the portion of the flow that transitions to the hardware layer. The red dashed line denotes the threshold θ at which this transition occurs. I refer to this process as *horizontal separation*.

5.1.4 Refining Observability of LAN Delays

The θ parameter does not guarantee a consistent number of observable LAN-side delays. Due to the vertical separation process, the first θ packets will have a substantially lower number of LAN delays, as WAN-side delays are excluded and packet bursts may occur, further reducing the useful PIAT values.

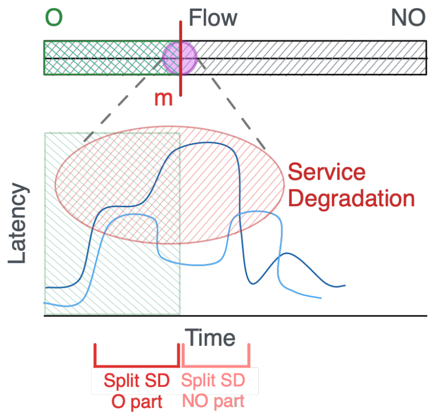


Figure 5.3: Illustration of a split SD event in a flow.

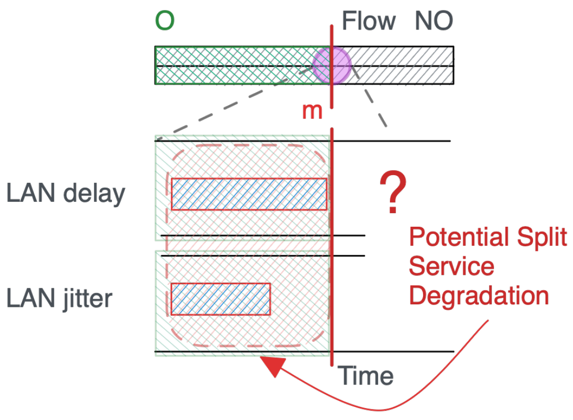


Figure 5.4: Illustration of a potential split SD event.

To focus on studying the behavior and predictive power of selected LAN delays, I introduce an additional threshold parameter, m, specifically for LAN delays. This threshold is defined similarly to θ but is applied exclusively to LAN delays. Essentially, the observable part of the flow consists of the delays measured up to this new threshold m, while delays collected beyond this point are considered non-observable. Similar to the packet-based separation, if a flow has fewer delays than the threshold m, it will be entirely observable.

5.1.5 Handling Split SD Events

While separating delay and jitter values based on the observable (O) and non-observable (NO) split is straightforward, identifying and handling SD events in this context is more complex. There are three possible scenarios when performing horizontal separation regarding SD events:

- The entire SD event falls into the O part of the flow (i.e., $SD_{\text{end}} < O_{\text{end}}$).
- The entire SD event falls into the NO part of the flow (i.e., $SD_{\text{start}} > O_{\text{end}}$).
- The SD event is split between the O part and the NO part of the flow (i.e., $SD_{\text{start}} < O_{\text{end}} \textit{ and } SD_{\text{end}} > O_{\text{end}}$).

In the latter case, it is necessary to decide whether to eliminate the SD event or to divide it into two smaller events. The rationale for eliminating the SD event is to prevent incomplete SD events from appearing at the end of the O part and the beginning of the NO part, especially when these segments do not meet the SD event requirements, such as being shorter than the MSL for that specific category. Conversely, the rationale for retaining and splitting the SD event is based on the importance of preserving potentially crucial information. Therefore, I chose to preserve these events by splitting them and marking them as split SD events for further study. Figure 5.3 visually illustrate this scenario.

However, this approach introduces another complexity: *how to handle LAN delay sequences that nearly qualify as SD events but fall short at the end of the O part?* These sequences resemble split SD events but do not develop into full SD events in the NO part. From an objective perspective, we can only observe the O part of a flow and lack information to determine whether a partial SD event is a true split event. To address this, I mark

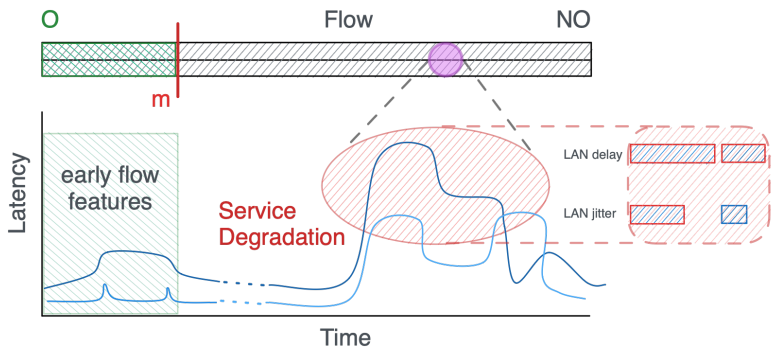


Figure 5.5: Intra-flow SD detection.

both actual split SD events and apparent split SD events, where delays end precisely at the O part's boundary, as split SD events. This scenario is depicted in Figure 5.4.

Additionally, I calculate the ratio of the length of a split event to the MSL for that flow's application category as follows:

$$f_{\text{split SD ratio}} = \frac{f_{|\text{partial SD event}|}}{f_{\text{MSL}}},$$

where f stands for a flow. The $f_{\text{split SD ratio}}$ may exceed 1 if the SD event at the end of the flow surpasses the MSL limit. This can occur when an SD event concludes exactly at the end of the flow or continues into the NO part of the flow.

Another type of split I record is the *apparent split*, which occurs when a real O delay ends precisely at the end of the O part, creating the illusion that it may be a split SD event continuing into the NO part.

5.1.6 Intra-Flow Service Degradation Detection

Preserving a high-quality user experience hinges on the timely and accurate detection of service degradation, a task that presents challenges in edge devices (such as home routers) with limited resources. Once the flow crosses the threshold into the non-observable domain, degradation becomes imperceptible.

The cornerstone of my method capitalizes on the observable states of specific flows. Using early flow features available up to the O/NO split threshold, I make educated inferences about the behavior and characteristics of flows that have advanced into the non-observable realm. This strategy involves harnessing the wealth of information from the observable pool to infer the state and performance of flows that elude direct observation. I call this procedure *intra-flow service degradation detection* as it uses information available within a specific flow to predict later flow behavior, hence the intra-flow nature of the detection. Figure 5.5 visually depicts this process.

The ultimate objective is to perform a binary classification using information from the observable portion of the flow to determine the presence or absence of service degradation in the non-observable portion. By analyzing the early flow characteristics, my goal is to accurately predict whether an SD event occurs later in the flow.

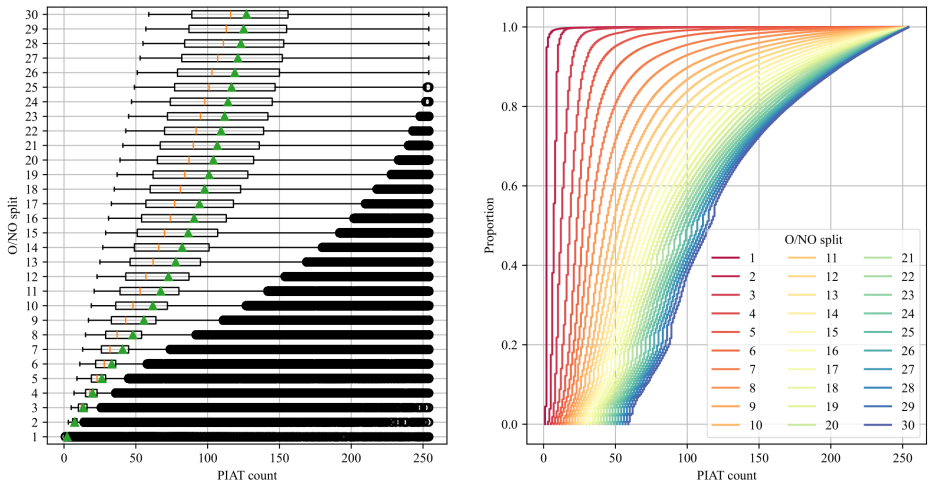


Figure 5.6: Relationship of O/NO split thresholds of LAN delays (m) to the O/NO split threshold of packet (or PIAT) counts (θ)

5.2 Analytical Framework

5.2.1 Threshold Analysis for Effective LAN Delay Monitoring in SD Detection

In this study, I aim to identify an optimal m value that allows for accurate determination of flow behavior based on the O part, thereby enabling high SD detection levels in the NO part. However, the parameter that can be influenced in a production environment is the θ threshold. To derive this for a specific O/NO split threshold, I adopt an empirical approach. During the vertical separation of each flow, I record the actual PIAT count leading to a specific number of LAN delays. By analyzing the distribution of PIAT counts for a specific O/NO split threshold, we can estimate the number of PIATs needed to capture the required number of LAN delays, thus determining the corresponding θ threshold.

Figure 5.6 shows the boxplot for each m value on the y-axis and the distribution of PIAT counts (θ) on the x-axis in the plot on the left. The plot on the right displays the corresponding ECDF plots. I observe that between O/NO splits of 1 and approximately 16, the PIAT count ranges consistently widen. Beyond this point, the range of possible PIAT counts remains relatively stable, although the center shifts. The median PIAT count is about four times the LAN delay count, indicating that we need to collect approximately four times the PIAT values for a specific m in around half of the flows. For instance, if the O/NO split is set at 10 LAN delays, we need to monitor about the first 40 packets of the flow to achieve this count.

Table 5.1 presents the relative PIAT count for each O/NO split at 80, 85, 90, 95, and 99 percent on the ECDF plot. The table shows the ratio of collected PIATs relative to LAN delays for the corresponding percentage of flows.

Table 5.1 presents the relative PIAT count for each O/NO split for 80, 85, 90, 95, and 99 percent on the ECDF plot. The table shows the ratio of collected PIATs relative to LAN delays that is valid for the corresponding percentage of flows. From Table 5.1, I find that to ensure at least m LAN delays (i.e., a fully observable part) for about 80 percent of the flows, we need to collect at most 5-8 times as many packets. At 90 percent, this ratio exceeds 10 times the LAN delay count in some cases and remains above 7 times for the majority of examined O/NO splits.

Table 5.1: Relative PIAT count for a specific m O/NO split threshold shown for a specific ECDF proportion

m	**80%**	**85%**	**90%**	**95%**	**99%**
5	6.20	6.60	7.60	10.00	18.80
10	8.10	9.40	11.50	15.30	22.00
15	8.00	9.13	10.80	13.13	16.00
20	7.30	8.20	9.30	10.80	12.30

It is important to note that a twofold ratio is expected since I am only examining LAN delays, leaving at least half of the communication unexamined. To examine at least 10 LAN delays for 80 percent of the flows, we need to observe 81 packets and the corresponding PIATs on the software side. Conversely, if we consider only 5 LAN delays, we need around 31 packets.

The relationship between LAN delay counts and packet counts underscores the challenge of determining an optimal threshold. I aim to identify the lowest m threshold that provides a satisfactory view of flow behavior while minimizing the impact on high-speed networking, even when translated to the θ threshold.

5.2.2 Statistical Analysis of Optimal O/NO Split Threshold

To empirically estimate the optimal O/NO split threshold, I perform a statistical analysis of LAN delays. The goal is to identify common patterns in the behavior of LAN delays that could suggest a universal m threshold. I conduct this analysis in an application type-agnostic manner to identify a single threshold suitable for all application types.

One heuristic that can indicate a common O/NO split threshold is the behavior of cumulative LAN delays. If we observe constant increases for the majority of flows, the split point is arbitrary since the delay behavior does not change throughout the flow. Thus, the O part's behavior reflects what we would expect in the NO part. Conversely, if cumulative LAN delays change at certain points, including these changepoints in the observable part could provide a better understanding of flow behavior. When examining the cumulative LAN delay of flows at specific LAN delay counts, we see that many flows exhibit a linear increase in cumulative delay. However, there is significant variability among the flows: some have steeper increases, while others have more gradual ones. Common inflection points, where multiple flows experience changes in their cumulative LAN delay growth, suggest these points may indicate general patterns.

To further study changepoints, I compute them for all flows using the CUMSUM method. This method accumulates the cumulative sums of LAN delay changes (essentially cumulative jitter) of each flow as a time series in both positive and negative directions. The accumulators are offset (decreased for the positive accumulator and increased for the negative one) by a 5 ms drift to reduce the impact of LAN delay changes. Once an accumulator reaches an arbitrarily set threshold of 10 ms (or -10 ms), that index is marked as a changepoint.

From the distribution plot of changepoints in Figure 5.7, I observe certain LAN delay indices where many flows experience changes. This could indicate common network events or patterns impacting multiple flows simultaneously. From Figure 5.7, I find that the vast

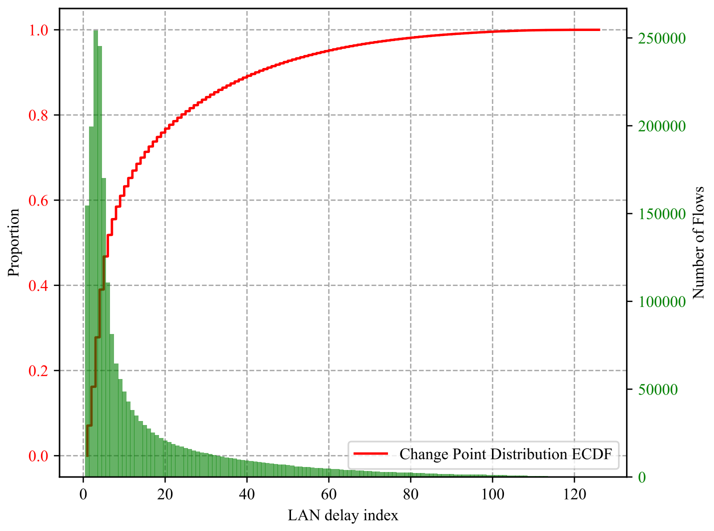


Figure 5.7: Distribution of changepoints across all flows.

majority (close to 80% from the ECDF plot) of changepoints occur before the 20th LAN delay, with the highest concentration around the 5th delay. This suggests a potential selection range (between 5 and 20 LAN delays) for the O/NO division point, as the early part of the flow experiences more volatility. The ECDF plot also indicates that to cover around 90% of the changepoints, I would need to double the observed delays. Thus, a higher LAN delay count may result in diminishing returns for changepoint identification. Based on these findings, I will examine analysis results for O/NO splits at 5, 10, 15, and 20 LAN delays. These splits cover the identified range of changepoints well and offer insights into O/NO flow behavior at granular steps.

The average number of changepoints detected for each flow is approximately 1.90. This gives us an understanding of how often, on average, there is a notable change in the delay characteristics of a flow. Despite the average being 1.90, the range (from 0 to 120 as depicted in previous plots) suggests high variability among flows. This variability underscores the dynamic nature of flows, with significant changes occurring during their duration.

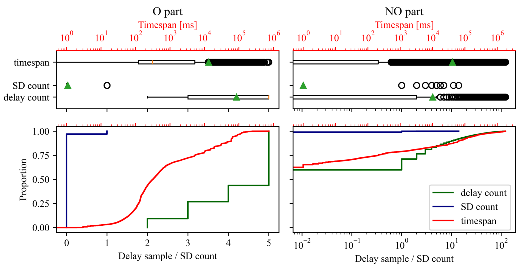
5.2.3 Flow Feature Analysis Following the O/NO Split

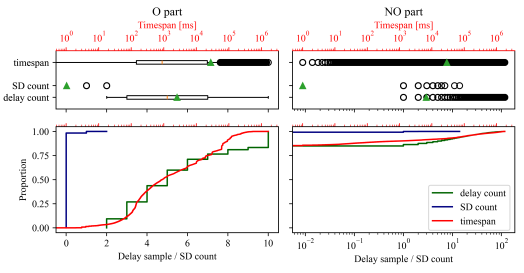
In this section, I analyze the behavior and distribution of different flow features following the O/NO split at the identified thresholds.

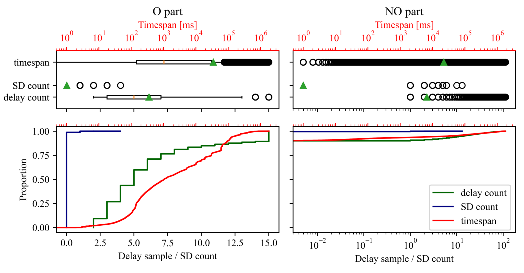
5.2.3.1 Delay Count, SD Count, and Timespan O/NO Distribution

Table 5.2: Selected O/NO Splits Data

m	**O delay%**	**O SD%**	**SD NO/O% pred.**	**avg. $f_{\text{split SD ratio}}$**
5	50.07	2.32	50.64	0.007280
10	65.34	47.55	32.55	0.010874
15	72.88	77.09	32.29	0.007449
20	78.14	86.03	39.83	0.004398


(a) O/NO split: 5


(b) O/NO split: 10


(c) O/NO split: 15

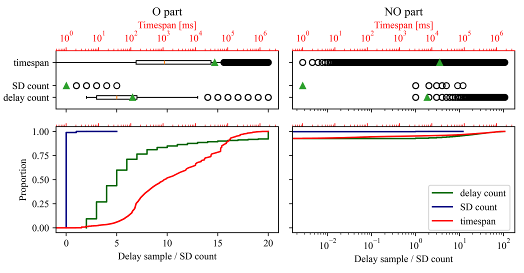

(d) O/NO split: 20

Figure 5.8: Delay count, SD count and timespan O/NO distribution at different O/NO split thresholds

Figure 5.8 presents the distribution of delay count, SD count, and timespan between the O and NO parts for O/NO splits of 5, 10, 15, and 20. As the O/NO split threshold increases, the O parts of the network flows capture a longer timespan, corresponding to an increase in delay counts and SD counts. The majority of flows do not exhibit SD events in either the O or NO parts, as indicated by the mean of 0 in the boxplots and the vertical step in the ECDF plots at this count. For flows with SD events, the maximum number of SD events in the O part rises from one at an O/NO split of 5, to two at a split of 10, four at 15, and five at 20. In the NO parts, the distribution of SD event counts remains consistent across thresholds, ranging from 1 to 15.

The delay counts in the O part align with the expected distributions, with delays reaching up to the thresholds. The median delay count remains at 5 across all thresholds, with the mean also around this value. ECDF plots indicate that approximately 70% of flows have at most 5 delay counts at thresholds from 10 to 20, showing that most flows are short and that the O parts cover a significant portion even at a split of 5. In the NO parts, over 80% of flows have at most 10 delays, and at threshold 5, more than half of the flows have a maximum of one delay. Most flows have no delays in the NO part at other thresholds, but outliers with over a hundred delays skew the average delay count to 2 at all threshold levels.

The timespan also exhibits similar patterns to delay counts, with an average increase of 10 seconds per threshold level. At threshold 5, the mean timespan is around 10 seconds. However, the overall distribution of timespans is similar across all thresholds, with comparable ECDF plots, similar boxplot extents, and median values. The median timespan is 1 second for thresholds 10 to 20, and 300 ms for a split of 5. In NO parts, the timespan can extend up to 30 minutes, the active timeout for expiring flows. At threshold 5, 70% of flows have NO parts of 1 ms or less, increasing to around 80% at thresholds 10 and 15, and over 90% at 20. This indicates that higher O/NO splits capture significantly fewer flows in the NO part, except for substantially longer flows.

Table 5.2 presents numerical statistics regarding the delay and SD distribution across the O and NO parts of the flows. *O delay%* measures the percentage of delay falling into the O part (implying the ratio in the NO part). *O SD%* does the same for SD events. *SD NO/O% pred.* is the percentage of flows that have SDs in the O part and NO part too, hinting at the predictive value of this metric alone for NO SD classification. The average $f_{split\ SD\ ratio}$ is also shown.

The share of SD in the O part for an O/NO split threshold of 5 is only 2.32%. This increases steadily, reaching almost 50% at threshold 10, over 77% at 15, and over 86% at threshold 20. The delay count shows a similar pattern, starting at 50% and reaching 78% of delays in the O part by threshold 20.

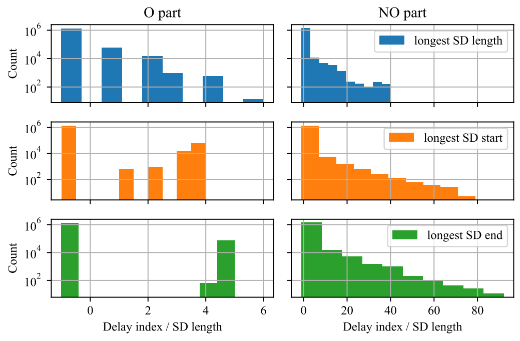
The SD NO/O% pred. is at 50% at an O/NO split threshold of 5, meaning that only half of the flows that have SD in the O part also have SD in the NO part. At threshold 10, it drops to only 32%, remaining at the same level at threshold 15, with an increase to 39% at threshold 20. This metric fluctuates for different O/NO splits, explained by the split SDs and guessed split SDs. When an SD event in the NO part at lower thresholds gets split into the O part, it generates an SD event in the O part even if there was no SD previously, influencing this metric. Similarly, if outlier delays that were not long enough to trigger an SD event reach the end of the O part, this triggers a guessed split SD event as well. This behavior is not included in the calculation of the SD O% metric (as it may result in higher than 100% of the delays occurring in the O part) but is captured by the SD NO/O% pred. metric, which examines how well such a simple heuristic predicts the presence of SD in the NO part of the flow. This indicates that this metric alone yields poor prediction performance. In other words, not all flows that have SD in the O part have SD in the NO part.

5.2.3.2 Characteristics of the Longest SD Event in the O Part and NO Part

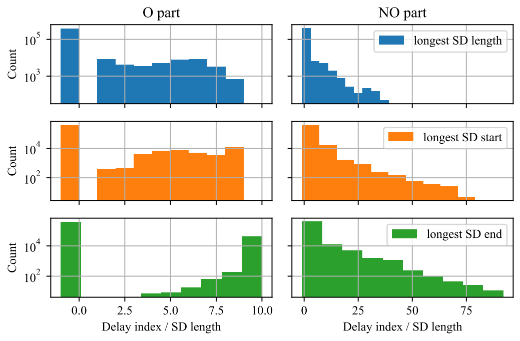
Analyzing the length, start, and end indices of the longest SD event in both the O part and NO part of the flow provides valuable insights for predicting similar metrics in the NO part. These metrics are crucial for understanding the behavior of SD events and improving predictive models.

Figure 5.9 displays histograms of metrics related to the longest SD event in network flows for both O and NO parts, split by O/NO thresholds of 5, 10, 15, and 20. The top histograms show the length of the longest SD event, the middle histograms show the start index, and the bottom histograms show the end index of the longest SD event. The start and end indices are absolute, meaning the lowest possible NO index equals the threshold value (e.g., 10 for a threshold of 10). Spikes at -1 indicate flows without any SD events in the respective part (O or NO).

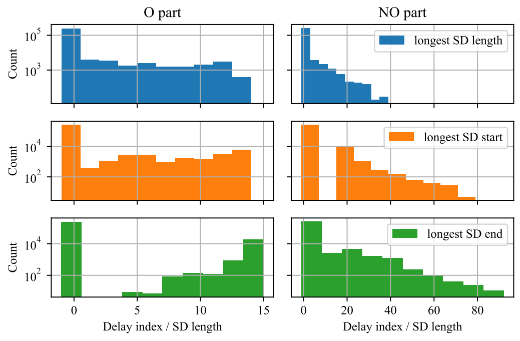
For the O part, the length of the longest SD event generally increases with higher thresholds. At a threshold of 5, most SD events are less than 2 delays long, although there are a few SD events lasting 3 or 4 delays. The number of SD events with a length of 5 delays is negligible. In contrast, at higher thresholds, the distribution of SD event lengths is more uniform. There are approximately the same number (around 1,000) of maximum length SD events that last for 1 delay as there are for those lasting 5, 10, or 19 delays. However, the vast majority of flows have no SD events at all, as previously discussed. In the NO part, all thresholds show a decreasing frequency of SD events as the length increases, with the longest SD events reaching around 40 delays for all thresholds.



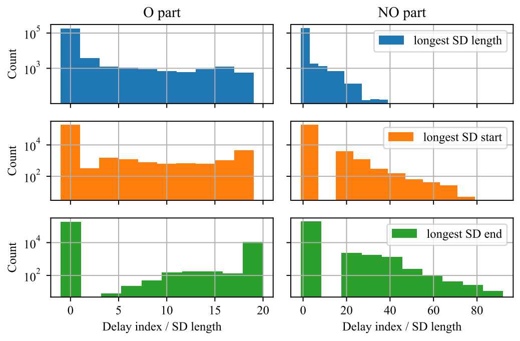
(a) O/NO split: 5



(b) O/NO split: 10



(c) O/NO split: 15



(d) O/NO split: 20

Figure 5.9: Characteristics of the longest SD event in the O part and NO part

The start index histograms for the O part reveal that at a threshold of 5, SD events mostly start before the fourth delay. For the other thresholds, SD events can start at any delay. In the NO part, the start indices range from the threshold value up to about 80 delays, with higher counts clustering near the lower threshold values.

The end index histograms show a similar trend. For the O part, the end indices show higher frequencies towards the end of the O portion, with the highest count at the threshold limit indicating either a split SD event or an apparent split. In the NO part, end indices are widely spread, showing patterns similar to the start indices, indicating that SD events can extend across a significant portion of the NO segment, reaching over 80 units.

5.2.3.3 Studying SD Location and Split SDs

Another important metric is whether an SD event was split between the O part and NO part, where the SD events are located in the O and NO parts, and the number of guessed and apparent splits. This section examines the cardinality of these metrics.

The two grouped bar plots in Figure 5.10 display the characteristics of SD events and network flows across different O/NO split thresholds (5, 10, 15, and 20). The left plot, labeled *SD stats*, counts all SD events separately, even if multiple events occur within a single flow. Categories include short O SDs, O SDs, apparent split SDs, real split SDs, guessed split SDs, and NO SDs. Short O SDs represent SD events within O parts shorter than the O threshold. O SDs are events entirely within the O part, while apparent split SDs touch the end of the O part, giving the illusion of split events. Real split SDs span both O and NO parts, whereas guessed split SDs are events that appear as the beginning

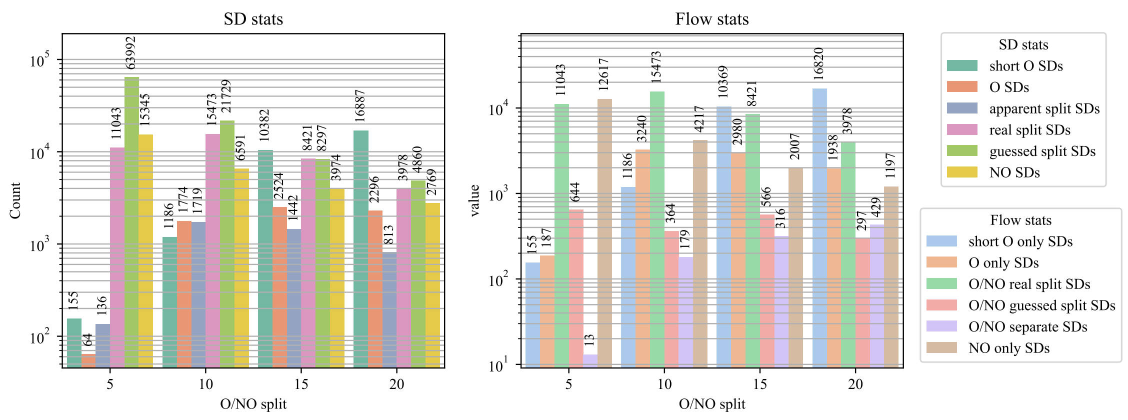


Figure 5.10: Location of SD events in the flows.

of an SD event at the end of the O part but do not grow into a real SD event in the NO part. NO SDs occur entirely within the NO part.

The right plot, labeled *Flow stats*, categorizes flows based on their SD events. If a flow has multiple SD events, it is counted in a single category. Categories include short O only SDs, O only SDs, O/NO real split SDs, O/NO guessed split SDs, O/NO separate SDs, and NO only SDs. Short O only SDs indicate flows shorter than the O part. O only SDs are flows with SDs exclusively in the O part, while O/NO real split SDs have confirmed split SDs. O/NO guessed split SDs include guessed splits, and O/NO separate SDs have distinct SDs in both parts. NO only SDs represent flows with SDs entirely in the NO part. Flows too short for NO part analysis are indicated with a patterned filling. Apparent splits here are regarded as normal O SDs and are counted either as O only SDs or O/NO separate SDs based on further flow behavior.

As I increase the threshold, the number of flows with fewer delays than the threshold increases. These flows are excluded from further analysis as they do not have an NO part. At threshold 5, the number of O SDs is low with a few additional apparent split SDs; however, the number of splits and NO SDs are orders of magnitude higher. The highest count is for guessed splits, exceeding 63,000. The balance between the metrics improves for higher thresholds, where the number of O SDs (including apparent splits) is in the same order of magnitude as the number of NO SDs. For all thresholds, splits and guessed splits have the highest cardinality. However, except for threshold 5, these are distributed more equally.

The flow analysis shows similar patterns, with the count of short O only SD flows increasing with higher thresholds, and flows with split SDs dominating. Here, the real splits clearly outnumber guessed splits (by an order of magnitude), indicating that this metric is crucial for predicting whether an NO part will have an SD event. The number of flows with SDs only in the NO part is similarly high but decreases steadily as the threshold increases. There are only a few flows with separate SD events in both the O and NO parts at threshold 5, while this number increases to match the number of flows with guessed splits at higher thresholds.

The average $f_{split\ SD\ ratio}$ tends to be very low for every threshold examined, as shown in Table 5.2. This means that only a small number of flows have split SDs, and those that do tend to be shorter than the MSL of the corresponding application category.

5.3 Model Evaluation and Performance Analysis

In this section, I explore the potential of the O parts of flows to predict various metrics in the NO parts. I aim to assess how well information from the O part can infer characteristics of the NO part. Specifically, I use classification models in a binary setting to determine the presence or absence of SD events in the NO part. Additionally, I utilize regression models to predict the number of SD events and various attributes of the longest SD event in the NO part, such as its length, and the start and end indices relative to the flow's beginning.

Table 5.3: Overview of Experimental Design

Objective	Null Predictor	All True Predictor	Random Predictor	O SD-based Predictor	Split SD Predictor	Logistic/ Ridge	XGBoost	MLP
Objective 1	✓	✓	✓	✓	✓	Logistic	✓	✓
Objective 2	-	-	-	✓	✓	Ridge	✓	✓
Objective 3	-	-	-	✓	✓	Ridge	✓	✓
Objective 4	-	-	-	✓	✓	Ridge	✓	✓
Objective 5	-	-	-	✓	✓	Ridge	✓	✓

5.3.1 Data Preparation

Data collected from Monday to Wednesday was used as training data, while data from Thursday and Friday was used for testing. Only flows that last at least until the O/NO split were included, as shorter flows do not have an NO part.

The input data includes the following features:

- Minimum, maximum, median, mean, and standard deviation of delays and jitters in the O part.
- Delay and jitter values in the O part as individual features.
- Count of SD events in the O part.
- Length, start, and end indices of the longest SD event in the O part, indexed from the start of the O part.
- Name of the application and application category of the flow.
- Location and connection type (wired or wireless) of the flow.
- $f_{\text{split SD ratio}}$.

The application name, application category name, location, and connection type columns were one-hot encoded to ensure compatibility with all models. Additionally, I created a scaled version of the input datasets using the Standard Scaler method, as the MLP algorithm requires scaled data.

Table 5.4 shows the input sizes as flow count and feature count for the training and testing sets for each O/NO split value. The discrepancy in test sizes as the O/NO split threshold changes is due to fewer flows meeting the criterion of running at least up to the end of the O/NO split threshold. Additionally, while all application categories, connection types, and locations are represented at each threshold, the observed applications differ slightly for the same reason.

Table 5.4: Train and Test Sizes with Input Feature Counts for Different O/NO Splits

O/NO Split	Train Size	Test Size	Input Feature Count
5	905,407 flows	507,359 flows	141 features (including 6 application category name, 101 application name, 9 location, 2 connection type OH encoded features)
10	264,183 flows	154,021 flows	140 features (including 6 application category name, 90 application name, 9 location, 2 connection type OH encoded features)
15	168,463 flows	99,290 flows	144 features (including 6 application category name, 84 application name, 9 location, 2 connection type OH encoded features)
20	122,814 flows	73,043 flows	152 features (including 6 application category name, 82 application name, 9 location, 2 connection type OH encoded features)

5.3.2 Experimental Design

This study aims to evaluate the predictive power of the O parts of network flows to predict various metrics in the NO parts. The modeling targets five distinct objectives:

- **Objective 1:** Determining the presence or absence of SD events in the NO part as a binary classification problem.
- **Objective 2:** Predicting the count of SD events in the NO part as a regression problem.
- **Objective 3:** Predicting the length of the longest SD event in the NO part, with a value of -1 if no event exists, as a regression problem.
- **Objective 4:** Identifying the start index of the longest SD event in the NO part, measured from the beginning of the flow, with a value of -1 if no event exists, as a regression problem.
- **Objective 5:** Identifying the end index of the longest SD event in the NO part, measured from the beginning of the flow, with a value of -1 if no event exists, as a regression problem.

To address these objectives, I trained various models for each target, as summarized in Table 5.3.

For objective 1, I evaluated a null predictor (always predicts false—no SD in the NO part); all true predictor (always predicts true—SD in the NO part); a random predictor; an O SD-based predictor (predicts true if there is an SD in the O part, otherwise predicts false); a split SD metric-based predictor (predicts true if the *split SD ratio* is greater than 0, otherwise predicts False); Logistic Regression; XGBoost; and Multi-Layer Perceptron (MLP).

For objective 2, I evaluated an O SD-based predictor (predicts 1 if there is an SD in the O part, otherwise predicts 0); a split SD metric-based predictor (predicts 1 if the *split SD ratio* is greater than 0, otherwise predicts 0); Ridge Regression; XGBoost; and MLP.

For objective 3, the models evaluated were an O SD-based predictor (predicts MSL if there is an SD in the O part, otherwise predicts 0); a split SD metric-based predictor (predicts MSL if the *split SD ratio* is greater than 0, otherwise predicts 0); Ridge Regression; XGBoost; and MLP.

For objective 4, the models used were an O SD-based predictor (predicts the index M if there is an SD in the O part, otherwise predicts -1); a split SD metric-based predictor (predicts the index M if the *split SD ratio* is greater than 0, otherwise predicts -1); Ridge Regression; XGBoost; and MLP.

Lastly, for objective 5, I evaluated an O SD-based predictor (predicts MSL + M if there is an SD in the O part, otherwise predicts -1); a split SD metric-based predictor (predicts MSL + M if the *split SD ratio* is greater than 0, otherwise predicts -1); Ridge Regression; XGBoost; and MLP.

I employed Grid Search to identify the best parameter settings for all models, and training was conducted using 5-fold cross-validation. For classification, I aimed to maximize the Area Under the Receiver Operating Characteristic (AUROC) for the Logistic Regression and XGBoost models, and minimize the training loss for the MLP. For the regression tasks, I optimized the negative mean squared error for both Ridge Regression and XGBoost. To ensure reproducibility, random states were set to 42, and the MLP was run for 1000 iterations. The parameters used for Grid Search across all models are detailed in Table 5.5.

Table 5.5: Parameters for Grid Search Across Different Models

Model Name	**Python Library**	**Parameters for Grid Search**
Logistic Regression	`sklearn.linear_model` [26]	'solver': ['liblinear', 'lbfgs'], 'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10], 'max_iter': [1000]
Ridge Regression	`sklearn.linear_model` [26]	'alpha': [0.1, 1, 10]
XGBoost Classifier XGBoost Regressor	`xgboost` [8]	'n_estimators': [100, 200, 500], 'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7]
MLP Classifier MLP Regressor	`sklearn.neural_network` [26]	'hidden_layer_sizes': [(50,), (100,), (50, 50)], 'activation': ['tanh', 'relu'], 'solver': ['sgd', 'adam'], 'alpha': [0.0001, 0.05], 'learning_rate': ['constant', 'adaptive'], 'max_iter': [1000]

5.3.3 Evaluation Metrics for Model Performance

5.3.3.1 Classification Metric

To compare the performance of different models for predicting the presence of SD events in the NO part, I used various evaluation metrics widely employed in related work. These metrics include *precision*, *recall*, and the *F_1-score* (the harmonic mean of precision and recall). While precision is the ratio of true positive flows to all flows identified as positive (i.e., having SD in the NO part), I also included the *Negative Predictive Value (NPV)*, which is the counterpart for flows that lack SD events. *Recall* (also known as sensitivity or True Positive Rate (TPR)) measures how well the model identifies positive flows, i.e., the ratio of true positives to all actual positives. The *specificity* (True Negative Rate (TNR)) is the equivalent metric for negative samples. *Accuracy* (the ratio of correct predictions to all samples) and *balanced accuracy*, which accounts for class imbalance by averaging TPR and TNR, were also included.

5.3.3.2 Regression Metrics

To comparatively evaluate the regression tasks, I used the *mean absolute error (MAE)*, *root mean squared error (RMSE)*, *median absolute error (Median AE)*, *mean absolute percentage error (MAPE)*, and R^2 metrics. MAE measures the average absolute magnitude of the errors compared to the target variable (SD count, length, start index, and end index of the longest non-observable SD event), making this metric directly comparable with the distribution of the corresponding metrics in Section 5.2.3. RMSE, derived from the mean squared error, represents the average squared difference between the estimated value and the actual one, transforming it back into the scale of the target variable. Due to the squaring of errors, RMSE is more sensitive to outliers than MAE. Median AE focuses on the median error, ignoring outliers and providing a representative error achieved by the model. The R^2 metric indicates the goodness of fit, showing how well the predictions approximate the actual data points, with $R^2 = 1$ representing perfect prediction. This metric also indicates how well unseen samples may be predicted by the model relative to the mean of the observed data. While R^2 typically ranges between 0 and 1, negative values are possible, indicating that the mean of the data provides a better fit than the predicted values. MAPE, similar to R^2, normalizes the error and makes it comparable across different models using a percentage value. However, MAPE is unusable if the target variable is zero in most cases, as it approximates infinity.

All classification and regression metrics can be found in the Scikit-Learn documentation[1].

5.3.4 Classification Performance

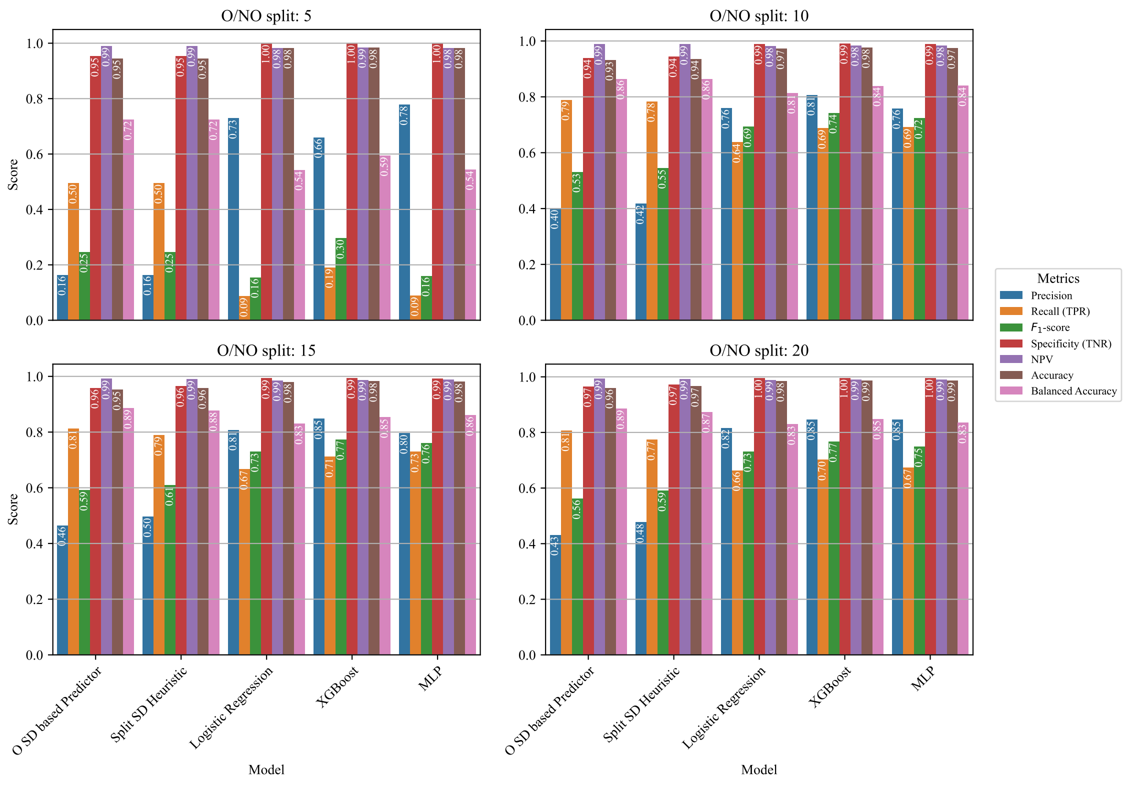


Figure 5.11: Classification metrics for NO SD presence prediction at different O/NO split thresholds.

Figure 5.11 compares the classification metrics for each O/NO split threshold studied, whereas Figure 5.12 depicts the Receiver Operating Characteristic (ROC) curves and corresponding areas under the curves (AUROC). From the figures, I find that all trained models showed significantly better performance than the random predictor, as well as

[1] https://scikit-learn.org/stable/api/sklearn.metrics.html#module-sklearn.metrics

the all-true or all-null predictors (not shown in the plots). The null predictor achieved an accuracy over 95% for all metrics (but only a balanced accuracy of 0.5), indicating a strong presence of flows with no SD events in the NO part. In terms of specificity, NPV, and accuracy, all models at every threshold showed excellent results (over 95% for all metrics, many approaching 100%). This suggests that all approaches are able to correctly classify flows that have no SD in the NO part, as these flows seem to exhibit consistent behavior with no SD events in the O part, nor split SDs (as demonstrated by the good performance of the simple heuristics). This is true both in terms of identifying the majority of such flows (high TNR) and having a high number of correct predictions for flows predicted not to have SD in the NO part (high NPV). The similarly high accuracy underscores the high imbalance towards these types of flows in the data. However, when it came to correctly identifying flows with non-observable SDs, the models did not perform as well.

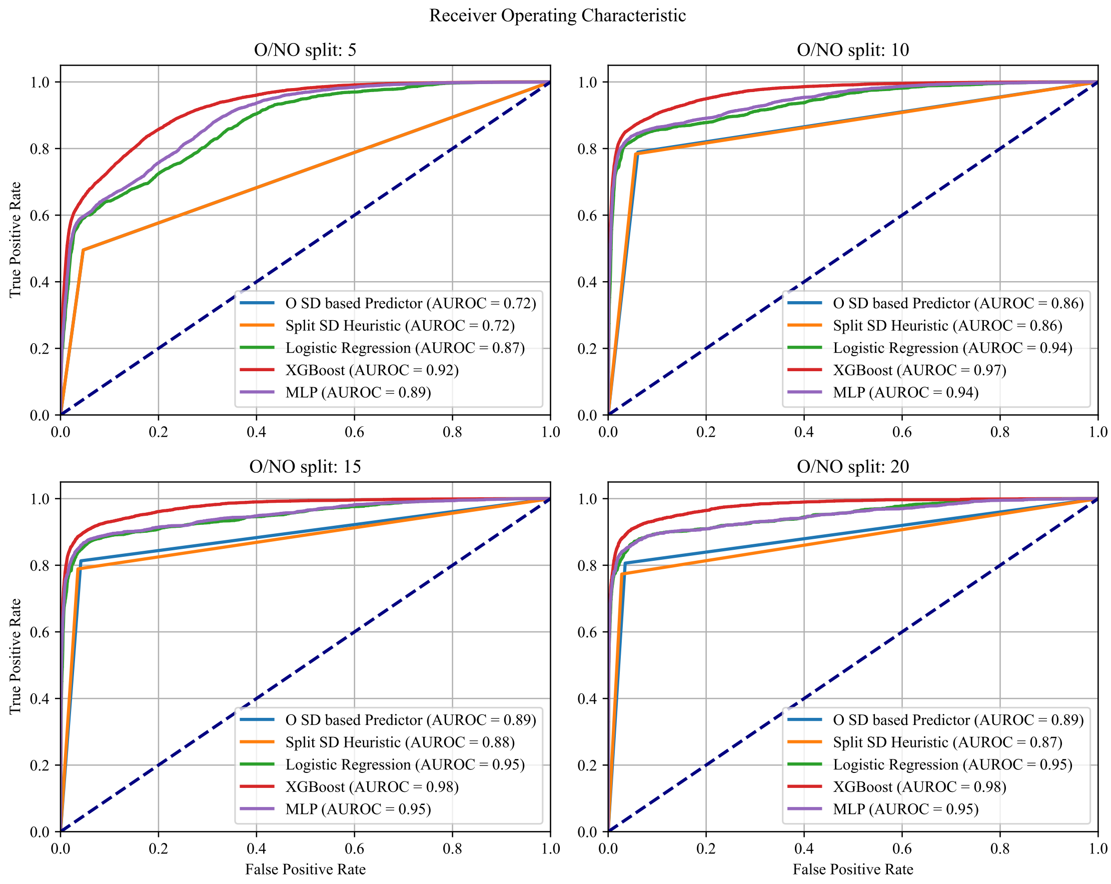


Figure 5.12: ROC curves with corresponding AUROC metrics for NO SD presence prediction at different O/NO split thresholds.

Heuristic models (SD Based Predictor and Split SD Heuristic) demonstrated moderate effectiveness with highly similar prediction results across all metrics when examined individually at each threshold. They achieved the highest recall and balanced accuracy at all O/NO split thresholds (50% and 72%, respectively, for the two metrics at threshold 5, and around 0.8 and over 0.85, respectively, for the other thresholds). This indicates that among all models, they were able to identify the highest number of flows with SD events in the NO part, although still missing half of the positive flows. The high accuracy reflects this result, and the balanced accuracy is improved by the relatively high recall. However, these heuristics underperformed in terms of precision and, consequently, the

F_1-score when compared to other models. These results suggest that many of the flows with non-observable SD events also have SD in their O parts. However, many flows with SD events in the O part or guessed split SD events do not have SD events forming in the NO part. Correctly capturing these flows appears to require considering other early flow features not used by these simple heuristics.

In line with this, all sophisticated models, i.e., the Logistic Regression, XGBoost, and MLP, exhibited much higher precision at the cost of lower recall. This means that they identified fewer flows with SD in the NO part overall, but when they did, the prediction was more often a true positive. The discrepancy between these two metrics was greatest at the O/NO split threshold of 5, with XGBoost achieving a precision of 66% and over 72% for the other two models, while recall was only 19% for XGBoost and just 9% for Logistic Regression and MLP. Consequently, the F_1-scores were low, with 30% for XGBoost and half of this for the other two models, and a balanced accuracy of around 55-60%. This suggests that at this split threshold, the models did not gather enough information to confidently predict the more volatile behavior in the NO part.

By the O/NO split threshold of 10, the prediction metrics became more balanced for all models. The simple heuristics maintained their lead in recall (78%) and balanced accuracy (86%), while also improving their precision. However, precision remained under 45%, highlighting the superiority of the more sophisticated approaches, which offered balanced results with all metrics exceeding 64%. While recall was the lowest metric for these models, they approached the recall of the simple heuristics, especially XGBoost and MLP, which achieved a recall of 69% and a corresponding balanced accuracy of 84%. Their precision, however, remained considerably higher than that of the simple heuristics, with XGBoost excelling at over 80%. In terms of F_1-score, this model achieved the best result at 74%, highlighting its balance between recall and precision.

Going beyond this threshold to O/NO splits of 15 and 20 offered only marginal improvements for all models, with the previously observed patterns remaining consistent. The sophisticated models provided comparable results, with the XGBoost model excelling in terms of precision (85%) and F_1-score (77%). In balanced accuracy (85%), it fell short by one percentage point from MLP and by 4% from the O SD-based heuristic. At threshold 20, there were only minor improvements in specific metrics, with some even decreasing. This is explained by the significantly reduced number of flows considered, as many flows fall entirely in the O part at this threshold (about half the amount available for the O/NO split threshold of 10, as shown in Table 5.4). The results suggest that choosing such an elevated split threshold offers no clear benefits.

The ROC curves offer model performance explainability by varying the threshold between the positive (flows with non-observable SDs) and negative predictions. This provides a more comprehensive view of a specific model and allows for comparison of the robustness of different models. The curves show similar behavior to the patterns identified earlier. However, here the simple heuristics demonstrate clear inferiority to the other models, even at the O/NO split threshold of 5, with AUROC scores of 72% compared to over 87% for the other models. The superior robustness of the XGBoost model is evident at every threshold. At the O/NO split of 5, it already exceeds 0.9 in AUROC, reaching 0.97 by threshold 10 and 0.98 by 15. Its ROC curve is the highest among the studied models and follows a smooth pattern with no perceptible dips, unlike other models. Logistic Regression and MLP exhibit similar patterns with highly correlated ROC curves and nearly identical AUROC values. The lack of significant improvement beyond threshold 10 is also evident in this figure.

In conclusion, for the classification task, the majority of the studied flows had no SD events in the NO part, which all models were able to identify with high accuracy. While the simple heuristics capture the core features of non-observable SD patterns, they fall short compared to more sophisticated models. The XGBoost model showed the most promising results, offering a balance of high precision and good recall with a high balanced accuracy, outperforming other studied approaches. Given that this model has the lowest training time of the three and does not require feature scaling, it is an ideal choice for the non-observable SD prediction task in this research.

In terms of O/NO split thresholds, I identified threshold 10 as the best in terms of prediction performance relative to the O/NO split ratio. This threshold offers decent prediction performance at a reasonable O/NO split point. A lower threshold does not include enough information for a reliable prediction, while increasing the threshold further leads to only marginal improvements in performance, resulting in diminishing returns.

5.3.5 Regression Performance

Figure 5.13 summarises the results achieved for all examined regression problems and fitted models. The different plots correspond to the four regression tasks examined, with the top-left plot referring to the non-observable SD count, and the rest to the longest non-observable SD metrics. The regression metrics of different models are grouped together as four vertical columns (models represented on the x-axis), each representing an O/NO split threshold. Each marker inside such a column represents a specific regression metric. The MAE, RMSE, and Median AE metrics are shown on the left axes on a logarithmic scale. MAPE is depicted as the first scale on the right of the plots, also scaled logarithmically for SD count but linearly for the other regression tasks. Finally, the R^2 value is shown on the rightmost vertical scale on a linear scale. This representation of all models and metrics across different O/NO splits allows for a comprehensive visual comparison for each regression task. Models with better performance will have higher R^2 values and lower values for all other metrics in the plots.

An interesting observation in the case of the SD count prediction and the longest SD length statistic is that at the O/NO threshold of 5, the models resulted in generally good metrics, except for R^2. However, at the next threshold, all regression metrics, including R^2, improved. From this point on, at every threshold, R^2 consistently increased while the other metrics decreased. This indicates that the best-performing model will either be at the O/NO split threshold of 5 or 20. For the other two tasks, the O/NO split threshold of 5 seems to be the best, maintaining its edge (except for the R^2 metric), while other thresholds maintain significantly higher metrics.

The observable SD-based heuristic and split SD heuristic models produced poorer results than the other regression models for all tasks. Their R^2 metrics were all negative, at -1.5 for the count statistics and below -0.4 for the other tasks, indicating that the test data could not be effectively evaluated using these heuristics. The two heuristics showed similar behavior at each threshold and regression task pairing with minor differences. For the SD count statistic, the split SD heuristic at threshold 20 performed best with an MAE of 0.04 and RMSE of 0.2, and a median AE of 0. Due to the target variable frequently being 0, the MAPE is in the 10^{14} range, indicating its irrelevance as it approximates infinity. For the length of the longest non-observable prediction, the heuristic methods had slight deviations in their values. The best heuristic achieved a MAE of 1.31 and RMSE of 2.06, with a median AE of 1. The MAPE shows that these values were 115% higher than the target on average, indicating poor predictive performance for the SD length metric using

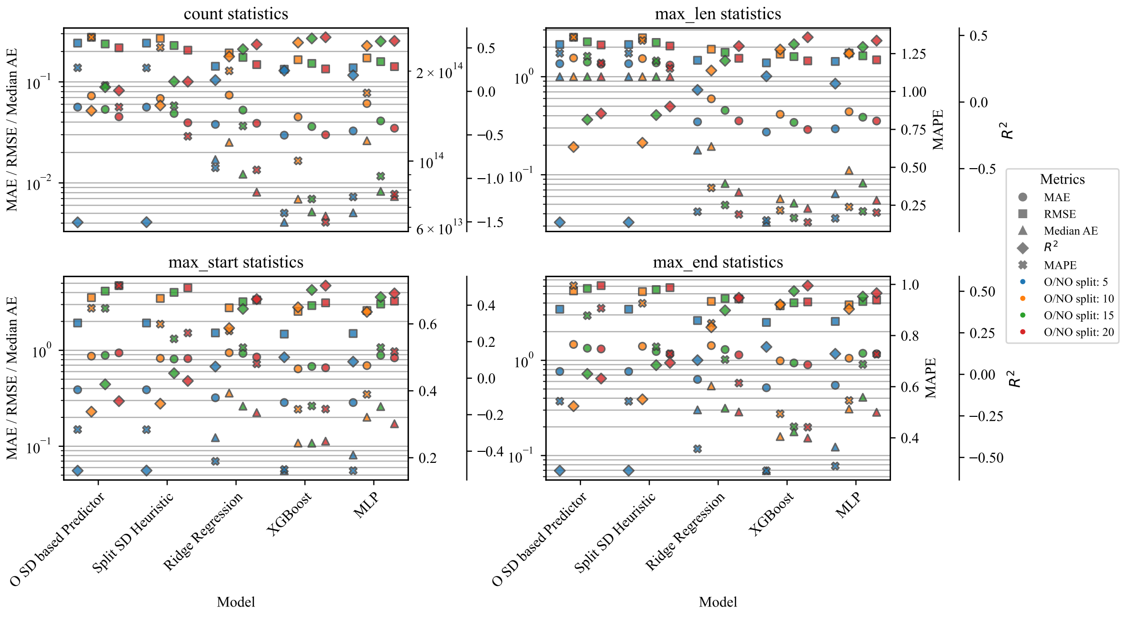


Figure 5.13: Regression metrics for NO SD count, longest SD length, longest SD start index and longest SD end index prediction at different O/NO split thresholds.

simple heuristics. When predicting the start and end indices of the longest SD event in the NO part, the heuristic models showed similar performance. Apart from the R^2 metric, the best threshold was 5, with the two models providing almost identical predictions. For the start index, the MAE was 0.38, MAPE 28%, median error 0; for the end index, the values were slightly higher, with an MAE of 0.76, RMSE of 3.44, MAPE of 54%, and median AE of 0.

Similar behavior was observed for Ridge Regression. While its metrics were generally more favorable than those of the simple heuristics, it still showed noticeably worse performance compared to XGBoost or MLP.

For the SD count prediction task, the best performing model was XGBoost. At the best threshold, it shares the O/NO split 5 and 20 as potential candidates with minimal differences. Threshold 20 takes the edge by having an R^2 of 0.62 vs. 0.23. The resulting model has a MAE of 0.03, RMSE of 0.13, and Median AE of 0.0047. Due to the target distribution, MAPE tends towards infinity for this model as well. The average errors being lower than the unit step (i.e., 1) of the target variable suggest that the model's predictions are very close to the actual values, with the error for at least half of the flows being extremely low (as indicated by the low median AE). The R^2 indicates a decent goodness of fit.

The length of the longest SD event in the NO part shares a similar scenario, with XGBoost yielding similarly good results at both O/NO split 5 and 20. Their MAEs are 0.272 and 0.289, RMSEs are 1.39 vs. 1.45, and Median AE are 0.032 and 0.045, respectively, while the MAPE is 15% for the former threshold and 14% for the latter. The main difference is again in R^2, with threshold 20 at 0.49 vs. 0.19 for threshold 5. The larger of these two still only indicates a moderate goodness of fit. The measured regression metrics suggest that models trained at both O/NO split thresholds perform well on this task, with the mean and median errors being considerably lower than the target unit.

For the start and end indices of the longest SD event, the clear best model (except for the R^2 metric) is XGBoost trained on data at O/NO split threshold 5. MLP at the same thresholds shares similar results as well. The metrics for the start index are 0.28 for MAE, 1.48 for RMSE, 16% for MAPE, 0.06 for median AE, and 0.11 for R^2. For the end index, the values are slightly higher, with a MAE of 0.51, RMSE of 2.50, MAPE of 27%, median AE of 0.07, and R^2 of 0.17. These values both show very poor goodness of fit, indicating that new samples may not fit the model well. However, as previously noted, the R^2 value may be negatively influenced when used for the evaluation of an algorithm that builds a sophisticated model. Focusing on the other regression metrics, both regression tasks showed good performance in predicting the target, with the prediction of the start index being slightly better.

The described results suggest that the best-fitted regression models (and even simpler heuristics) are capable of predicting the target variables well, with errors staying well below the unit steps of the target variables, hinting at a close-to-perfect predictive power. However, it is important to note that the presented metrics are all skewed when class imbalance is present. Since in this case there is a strong imbalance for flows with no SD events in the NO part of the examined flows, the observed results essentially describe the predictive power of the studied models for such flows (0 for non-observable SD count and the length of the longest SD event, while -1 for the indices). As observed in the classification task in Section 5.3.4, all models—including the heuristics—are highly capable of identifying flows without non-observable SD events, and when such a scenario was predicted, the results were mostly accurate. This behavior is reflected in the previously described regression metrics. Despite the error values being lower than the unit step, this is not necessarily the case for flows that actually have NO SD events, due to the imbalance of SD-lacking flows skewing the averages towards those flows. To eliminate the influence of these flows, I also plotted the same metrics incorporating only the flows that have SD events in the target set. The corresponding plots are shown in Figure 5.14. The metrics are encoded and presented in the same way as in the earlier figure, but here all axes follow a linear scale. I omitted showing the R_2 metric, as it was negative in all scenarios due to evaluating the metric on a subset of fitted samples.

As expected, the results in these cases are significantly inferior to the values observed earlier. The patterns in this limited analysis also differ greatly. Whereas the O/NO split threshold of 5 showed the most promising results when considering all flows, it was outperformed by most other thresholds in the majority of the studied metrics when focusing only on flows with non-observable SDs. However, it is important to note that different thresholds operated on SD events with distinct lengths, start, and end indices due to horizontal separation. Another distinction is that the simple heuristic models either achieved the best results or were on par with more sophisticated metrics.

For the non-observable SD count statistics, the best models were the two heuristic ones at threshold 10, with an MAE of 0.30, RMSE of 0.62, MAPE of 25%, and median AE of 0. This implies that at least half of the flows with NO SD events had only one SD event, as the heuristics only predict the presence of one anomaly. The low average error metrics indicate that the majority of the flows followed this pattern. Given that the range of SD event counts in the NO part for O/NO split threshold 20 is between 0 and 10 (as shown in Figure 5.8d), these errors are relatively low.

The length of the longest SD event task showed varying results. While the heuristic models at threshold 5 had the lowest MAPE, this was still at 68%, suggesting subpar predictive performance, and was paired with higher error metrics. The best model and threshold pairing in this case was XGBoost with threshold 15. Its MAE was 4.98, RMSE

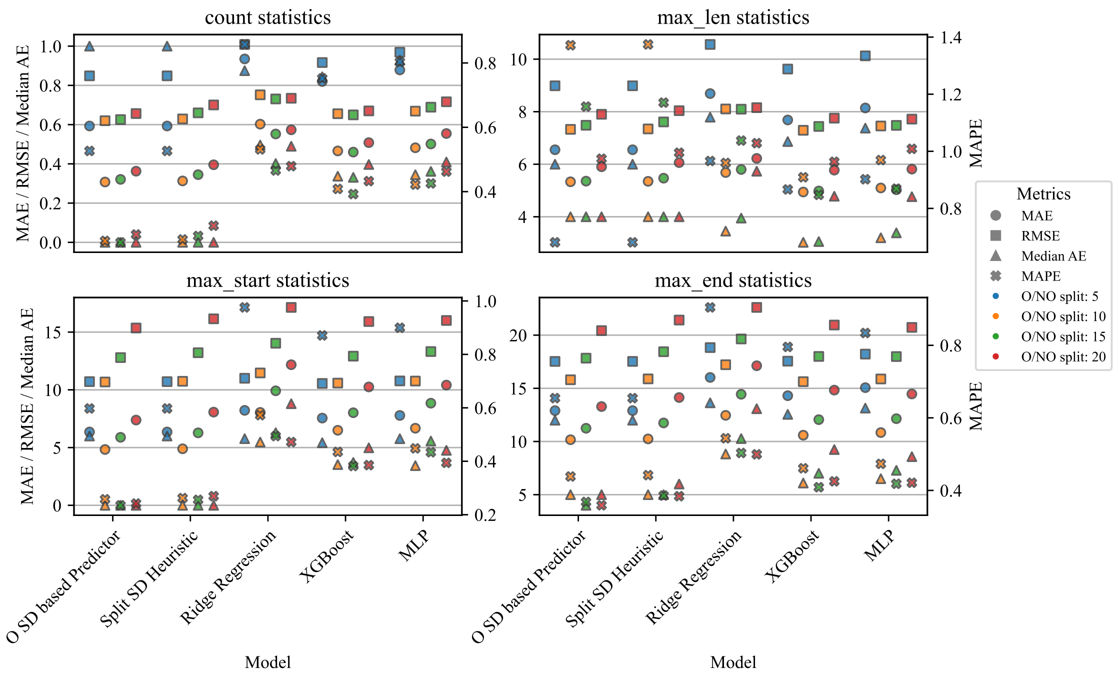


Figure 5.14: Regression metrics for NO SD count, longest SD length, longest SD start index and longest SD end index prediction at different O/NO split thresholds for flows that have SD events in the target set.

7.44, with a MAPE of over 84% and a median AE of 3.05. As shown in Figure 5.9c, the distribution of the length of the longest SD event is between 0 and 40 with a decreasing distribution. Therefore, the measured metrics are comparable to this range and thus may yield inaccurate predictions.

For the start and end indices, the heuristic models performed best again. The start index is best predicted by threshold 15 of either heuristic model. The MAE is 4.8, while the RMSE is 10.7. MAPE was measured at 26% with the median AE being 0. Like in the case of the SD count statistics, the heuristic model works well for at least half of the degraded flows, i.e., most flows have an SD event starting at the beginning of the NO part for this threshold. The case for the SD end index prediction is similar, with the observable SD-based heuristic taking the lead at threshold 10 for MAE and RMSE (at 10.16 and 15.80, respectively), and at threshold 15 for median AE and MAPE (4 vs. 5 for threshold 10 and 37% vs. 44%, respectively). XGBoost's prediction comes close to these models with a similar distribution but slightly higher metrics. Considering the distribution of the two targets (0 - 80 for start index and 0 - 85 for the end index, as shown in Figure 5.9), the predictions may be inaccurate.

In conclusion, for the regression tasks, when identifying flows with no non-observable SD events, all models perform well, with errors being lower than the unit step of all the target variables. The more sophisticated models take a slight edge over the heuristics in the studied metrics. However, when encountering flows with SD events in the NO parts, the prediction results are vastly more inaccurate. In these cases, the heuristics seem to perform better at predicting all metrics. This indicates that many of these flows have just one SD event that starts at the beginning of the NO part and lasts no longer than the MSL. However, the average error metrics measured were significantly higher and in the order of magnitude of the target variable, suggesting a high likelihood of incorrect predictions. The more sophisticated models were unable to capture enough information to predict more granular information about potential SD events in my analysis.

5.3.6 Feature Importance Analysis of the Best Classification Model

I analyse the best classification model to understand and identify the input features that had the highest weight in the decision. Only the classification task is examined, as none of the regression models in the examined tasks yielded satisfactory balanced results. To this end, I calculate the Shapley Additive Explanations (SHAP) values for the XGBoost model fitted on data following horizontal separation at threshold 10, which I previously identified as the best classification model.

In binary classification tasks, SHAP values provide a comprehensive measure of feature importance by indicating how much each feature contributes to the model's output for individual predictions. In this context, a positive SHAP value signifies that the feature pushes the prediction towards the positive class (i.e., flows with non-observable SD events), whereas a negative SHAP value indicates a push towards the negative class. The accompanying color gradient in SHAP summary plots, ranging from blue to red, represents the value of the corresponding feature, with blue denoting lower feature values and red denoting higher feature values. Purple colors mark the values that are close to the median or mean of the feature values.

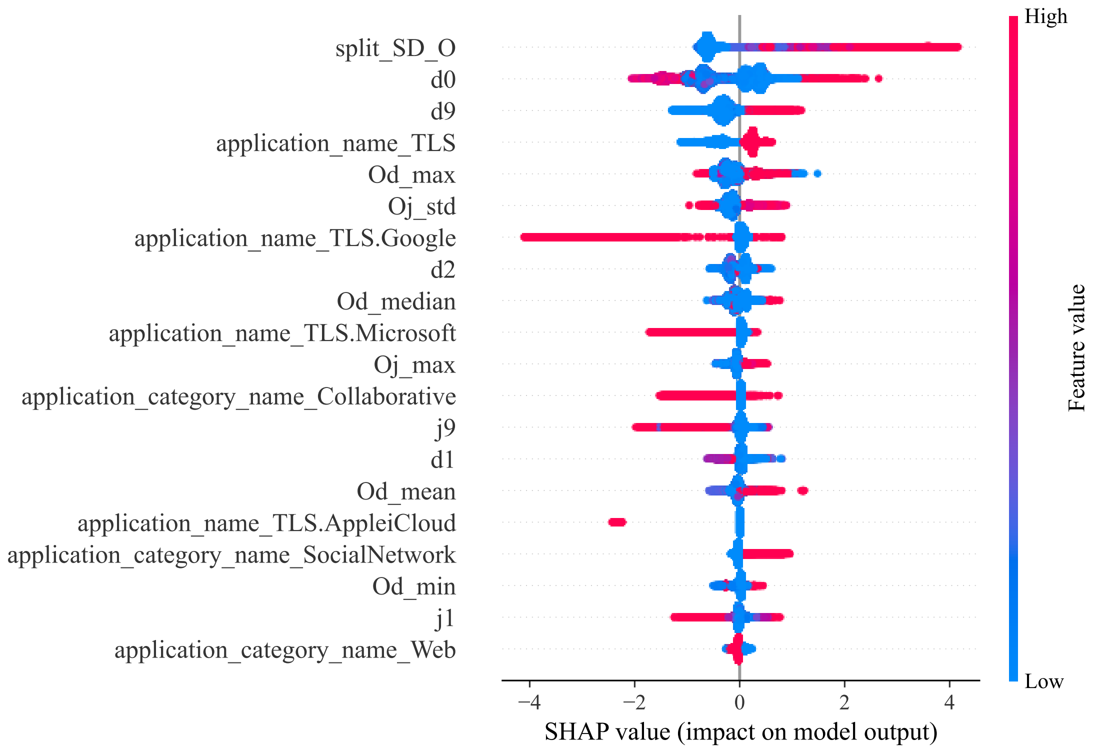


Figure 5.15: Classification SHAP values.

Figure 5.15 shows the SHAP values for the top 20 input features (from top to bottom) with the highest influence on the model decision in a summary plot. The most important feature for classification proved to be the $f_{split\ SD\ ratio}$, with higher ratios yielding higher SHAP values (between 2 and 4), thus increasing the likelihood of a positive prediction. In contrast, lower split ratios tend to correspond to negative SHAP values, incentivizing negative predictions, albeit with a much lesser impact (SHAP values of around -0.5). Moderate ratios also tend to increase the likelihood, but with lower SHAP values. This indicates that flows with a high $f_{split\ SD\ ratio}$ are likely to continue into the NO part as a split SD. The higher the ratio, the greater the impact on the decision, while low ratios (or 0) are less likely to have SD events. Since my heuristic model that uniquely used this value did not match the effectiveness of XGBoost, the latter model required additional features for better prediction.

The next top features are the first and last delay values in the O part of the flow, providing information from both the beginning and end of the flows. The SHAP values for the former show significant varied impacts. Low delays tend to have low SHAP values, both positive and negative, and do not significantly influence model decisions. High delays correspond to high SHAP values (between 1 and 2), while moderate delays have predominantly low SHAP values with similar impact (around -2). In other words, high starting delays increase the probability of a non-observable SD prediction, while moderate delays decrease it. The last delay sample in the observable part has a clear influence on the model output, with high delays pushing the model towards a positive prediction and lower values towards a negative prediction, albeit with a lower impact, as the corresponding SHAP values range between -1 and 1.

Beyond these, various application types influence the model prediction. Various TLS Collaborative applications seem to reduce the likelihood of a positive prediction, while the Social Network category tends to increase it slightly. The second highest absolute SHAP value corresponds to the `TLS.Google` application type, with strongly negative SHAP values (up to -4). Other aggregated delay and jitter features are also used, with higher values (for mean, max, standard deviation) typically having positive SHAP values, while low ones have negative SHAP values.

5.4 Discussion

5.4.1 Predictive Performance and Optimal Thresholds

My findings indicate that leveraging the observable portion of network flows to capture latency, jitter, and SD events, alongside the carried application type, demonstrates good predictive performance for identifying SD in the non-observable (NO) part. This research shows that analyzing 10 delay samples and other derived features strikes a balance between accurately predicting upcoming behavior and minimizing the number of packets processed on the slow path. While examining 5 delays showed subpar performance and increasing the threshold further led to only marginal improvements, the true optimal threshold for the O/NO split likely lies between these values. Further analysis is required to study the predictive performance between O/NO split thresholds 5 and 10.

5.4.2 Focus on LAN Delays

A key aspect of this study was the focus on LAN delays, as network providers cannot influence WAN behavior. I tested the predictive performance of LAN delays by splitting these delays into observable and non-observable parts. However, determining the best packet count for the O/NO split does not guarantee a consistent amount of LAN delays in the observable portion of the flows. To extract the required amount of LAN delays for at least half of the flows, we may need up to 4 times the captured packets, and up to

8 times for 80% of the flows. This translates to between 40 and 80 observable packets, which may exceed the resource capabilities of some devices or negate the benefits of fast path processing for short flows. Therefore, an important future direction is to examine the predictive performance of flows with a variable amount of delays when cutting at a strict O/NO packet limit.

5.4.3 Variable Delay Thresholds

My results also suggest that instead of a single packet threshold, it may be beneficial to keep the flow in the observable part until the required amount of delays are collected, then transition it to the fast path. This approach would mean some flows might only be examined up to their 20th packet, while others might require over 80 packets in the slow path. A comprehensive assessment is necessary to evaluate the resource utilization implications of this approach on the router. Additionally, studying the use of adaptive limits for flows with different characteristics, such as those belonging to distinct application categories, holds potential.

5.4.4 Model Performance and Limitations

In the experiments I execute, the XGBoost and MLP models were the best in extracting the necessary information for classification performance. While XGBoost can use raw data, MLP requires scaled input, which may skew the flow feature input out of the original range in production environments. Therefore, I prefer XGBoost, which also achieved slightly better performance in my testing. However, for the regression tasks, the models only matched the classification models' capabilities, reliably identifying flows without non-observable SD events. The heuristics performed better in predicting actual metrics for flows with SD events in the NO part, albeit with considerable error. Using classification and regression models in an ensemble might improve results, but additional information is likely needed to pinpoint SD length and start and end indices more accurately.

5.4.5 Dataset and Practical Considerations

This study used a dataset measured in a university dormitory environment with network traffic patterns indicative of home environments, suggesting potential transferability. If model retraining becomes necessary due to changing traffic patterns, it can be performed offline with a similar amount of captured flows. Model inference requires substantially fewer resources, making it feasible even for resource-constrained home network routers.

5.4.6 Future Directions

My results show that the $f_{split\ SD\ ratio}$ is a highly influential metric, indicating the presence of SD events at the beginning of the NO part. The trained model performs well in identifying these events but may miss others occurring beyond this point. To improve detection of these SDs and the accuracy of start and end index localization, we may need to harness additional information, such as the behavior of subsequent flows that have not yet transitioned to their respective NO parts. Tracking the delay and jitter behavior of these flows and their relative timing to the original flow may help better pinpoint SD events in the NO part, refining classification performance and improving regression metrics. This approach could also lower the required amount of LAN delay analysis, yielding a smaller optimal O/NO split threshold. Degraded flows could be redirected back to the slow path for potential QoS label application and preferential treatment by the CPU. My work provides a foundation for this potential future direction.

Chapter 6

Conclusion

This thesis has explored the identification and analysis of service degradation events using LAN data from a university dormitory setting, aiming to develop a methodology that could be applicable to a variety of network environments. My approach, centered on the detection of statistically significant deviations in network performance (specifically, extreme delays and jitter), has demonstrated potential applicability not only in similar institutional settings but also in residential and possibly enterprise networks.

The robustness of the methodology was validated by applying it across different locations and at different times, with results showing minimal deviation, thus underscoring the effectiveness of the SD event identification process. This consistency highlights the model's potential as a reliable tool for network administrators to preemptively address and manage network service quality issues. While my findings offer promising directions for network service management and SD detection, they also pave the way for subsequent studies to refine and expand on the groundwork laid here.

Future work considers extending the data collection period in non-academic residential settings to validate the method's efficacy in these contexts further. Exploring the scalability of the approach to accommodate larger or more complex network settings is also set as a future work.

In the second part of the thesis I focused on leveraging early flow features to predict service degradation in network flows after they transition from an observable state to a non-observable state, a method I call intra-flow service degradation detection. By capturing information from the initial part of a flow, I aimed to infer the behavior and characteristics of the flow after it moves to a fast-path processing but less observable state.

My analysis centered on the Packet Inter-Arrival Time metric, particularly those indicative of LAN side latency. I found that capturing the first 5 to 20 delays effectively covers most flow changepoints and encapsulates flow behavior. This range strikes a balance between gathering sufficient information and minimizing the number of packets processed on the slow path.

In evaluating my method, I tested simple heuristics and more sophisticated models for predicting SD events in the non-observable part of the flow. The XGBoost model emerged as the best performer, particularly at the O/NO split threshold of 10, achieving an F_1-score of 0.74, a balanced accuracy of 0.84, and an AUROC of 0.97. Key features influencing this model included the split ratio of an SD event and the first and last delay values in the observable part.

For regression tasks aimed at predicting specific SD-related metrics, my models were primarily effective in identifying flows without SD events in the non-observable part. However, they were less accurate for flows with SD events, highlighting the challenge of predicting more granular details.

The findings suggest that the optimal O/NO split packet threshold lies between 4 to 8 times higher than the delay count needed to capture the necessary delays. An adaptive threshold based on flow behavior could offer further improvements.

In summary, this thesis presents a statistical framework for identifying SD in network flows, and demonstrates the feasibility and effectiveness of using early flow features to predict SD in network flows after they become non-observable. Future work should focus on refining the adaptive thresholds, exploring inter-flow information for better SD event detection, and validating these methods in different network environments to enhance their robustness and applicability.

Acknowledgements

I would like to express my deepest gratitude to my two thesis advisors, Dr. Adrián Pekár for guiding me along the way, providing helpful feedback throughout the entirety of my thesis project, aiding me both in the access to the necessary resources and in refining my work. I am truly thankful for his friendly and flexible attitude towards me.

I would also like to extend my appreciation to my colleagues at work for being supportive, attentive and understanding during my thesis project.

I would also like to convey my earnest gratefulness towards my partner and family for supporting me in every aspect of my life and encouraging me during my studies.

List of Figures

List of Tables

Bibliography

[1] Atef Abdelkefi, Yuming Jiang, Bjarne Emil Helvik, Gergely Biczók, and Alexandru Calu. Assessing the service quality of an internet path through end-to-end measurement. *Computer Networks*, 70:30–44, 2014. ISSN 1389-1286. DOI: `https://doi.org/10.1016/j.comnet.2014.04.016`. URL `https://www.sciencedirect.com/science/article/pii/S1389128614001698`.

[2] Alex Amaral, Alejandra Armendariz, Santiago Erramuspe, Jose Joskowicz, and Mengying Liu. Prediction of video quality degradation on a cloud gaming platform. In *2023 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–5, 2023. DOI: `10.1109/BMSB58369.2023.10211223`.

[3] Zied Aouini and Adrian Pekar. Nfstream: A flexible network data analysis framework. *Computer Networks*, 204:108719, 2022. ISSN 1389-1286. DOI: `10.1016/j.comnet.2021.108719`. URL `https://www.sciencedirect.com/science/article/pii/S1389128621005739`.

[4] A. Bremler-Barr, E. Cohen, H. Kaplan, and Y. Mansour. Predicting and bypassing end-to-end internet service degradations. *IEEE Journal on Selected Areas in Communications*, 21(6):961–978, 2003. DOI: `10.1109/JSAC.2003.814614`.

[5] Dinis Canastro, Ricardo Rocha, Mário Antunes, Diogo Gomes, and Rui L. Aguiar. Root cause analysis in 5g/6g networks. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 217–224, 2021. DOI: `10.1109/FiCloud49777.2021.00039`.

[6] Pedro Casas, Pierdomenico Fiadino, Sarah Wassermann, Stefano Traverso, Alessandro D'Alconzo, Edion Tego, Francesco Matera, and Marco Mellia. Unveiling network and service performance degradation in the wild with mplane. *IEEE Communications Magazine*, 54(3):71–79, 2016. DOI: `10.1109/MCOM.2016.7432151`.

[7] Danilo Cerović, Valentin Del Piccolo, Ahmed Amamou, Kamel Haddadou, and Guy Pujolle. Fast packet processing: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):3645–3676, 2018. DOI: `10.1109/COMST.2018.2851072`.

[8] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. DOI: `10.1145/2939672.2939785`. URL `http://doi.acm.org/10.1145/2939672.2939785`.

[9] Vittorio Cortellessa and Luca Traini. Detecting latency degradation patterns in service-based systems. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 161–172, New York, NY,

USA, 2020. Association for Computing Machinery. ISBN 9781450369916. DOI: 10.1145/3358960.3379126.

[10] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622, 2014. DOI: 10.1109/IWCMC.2014.6906427.

[11] Fekadu Mihret Geremew Fanuel Melak Asmare and Lijaddis Getnet Ayalew. A survey on 3g mobile network traffic performance and analysis in ethiopia. *Cogent Engineering*, 9(1):2005506, 2022. DOI: 10.1080/23311916.2021.2005506. URL https://doi.org/10.1080/23311916.2021.2005506.

[12] FlowFrontiers. Service Degradation Detection - Dataset. https://github.com/FlowFrontiers/, 2024.

[13] FlowFrontiers. Service Degradation Detection - Digital Artifacts. http://github.com/FlowFrontiers/ServDeg-Dataset, 2024.

[14] FlowFrontiers. Service Degradation Detection - Dataset. https://github.com/FlowFrontiers/ServDeg-Intra, 2024.

[15] Christoph Hardegen, Benedikt Pfülb, Sebastian Rieger, and Alexander Gepperth. Predicting network flow characteristics using deep learning and real-world network traffic. *IEEE Transactions on Network and Service Management*, 17(4):2662–2676, 2020. DOI: 10.1109/TNSM.2020.3025131.

[16] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys Tutorials*, 16(4):2037–2064, 2014. ISSN 1553-877X. DOI: 10.1109/COMST.2014.2321898.

[17] Weigang Hou, Zhaolong Ning, Lei Guo, and Mohammad S. Obaidat. Service degradability supported by forecasting system in optical data center networks. *IEEE Systems Journal*, 13(2):1514–1525, 2019. DOI: 10.1109/JSYST.2018.2821714.

[18] Simon Kassing, Vojislav Dukic, Ce Zhang, and Ankit Singla. New primitives for bounded degradation in network service, 2022.

[19] Yunlong Li, Xinfeng Zhang, Chen Cui, Shanshe Wang, and Siwei Ma. Fleet: Improving quality of experience for low-latency live video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(9):5242–5256, 2023. DOI: 10.1109/TCSVT.2023.3243901.

[20] Feng Luo, Zitong Wang, and Baoyin Zhang. Impact analysis and detection of time-delay attacks in time-sensitive networking. *Computer Networks*, 234:109936, 2023. ISSN 1389-1286. DOI: 10.1016/j.comnet.2023.109936. URL https://www.sciencedirect.com/science/article/pii/S138912862300381X.

[21] Alessio Daniele Marra and Francesco Corman. From delay to disruption: Impact of service degradation on public transport networks. *Transportation Research Record*, 2674(10):886–897, 2020. DOI: 10.1177/0361198120940989.

[22] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. Hardware-accelerated network control planes. In *Proceedings of the 17th ACM Workshop on Hot Topics*

in Networks, HotNets '18, page 120–126, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450361200. DOI: 10.1145/3286062.3286080. URL https://doi.org/10.1145/3286062.3286080.

[23] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical Report RR-05-13, 2005.

[24] Lan N. Nguyen and My T. Thai. Network resilience assessment via qos degradation metrics: An algorithmic approach. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(1), mar 2019. DOI: 10.1145/3322205.3311072. URL https://doi.org/10.1145/3322205.3311072.

[25] T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 10(4): 56–76, 2008. ISSN 1553-877X. DOI: 10.1109/SURV.2008.080406.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Lucía Pons, Josué Feliu, Julio Sahuquillo, María E. Gómez, Salvador Petit, Julio Pons, and Chaoyi Huang. Cloud white: Detecting and estimating qos degradation of latency-critical workloads in the public cloud. *Future Generation Computer Systems*, 138:13–25, 2023. ISSN 0167-739X. DOI: https://doi.org/10.1016/j.future.2022.08.012. URL https://www.sciencedirect.com/science/article/pii/S0167739X22002734.

[28] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693, November 2015. URL https://www.rfc-editor.org/info/rfc7693.

[29] Nathan F. Saraiva de Sousa, Danny A. Lachos Perez, Raphael V. Rosa, Mateus A.S. Santos, and Christian Esteve Rothenberg. Network service orchestration: A survey. *Computer Communications*, 142-143:69–94, 2019. ISSN 0140-3664. DOI: 10.1016/j.comcom.2019.04.008. URL https://www.sciencedirect.com/science/article/pii/S0140366418309502.

[30] Luca Traini and Vittorio Cortellessa. Delag: Using multi-objective optimization to enhance the detection of latency degradation patterns in service-based systems. *IEEE Transactions on Software Engineering*, 49(6):3554–3580, 2023. DOI: 10.1109/TSE.2023.3266041.

[31] J.W.H. van Bloem and Roelof Schiphorst. *Measuring the service level in the 2.4 GHz ISM band*. Number TR-CTIT-11-27 in CTIT Technical Report Series. Centre for Telematics and Information Technology (CTIT), Netherlands, December 2011.

[32] P. Velan. Improving network flow definition: Formalization and applicability. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5, 04 2018. DOI: 10.1109/NOMS.2018.8406203.

[33] Hua Wu, Ya Liu, Shanshan Ni, Guang Cheng, and Xiaoyan Hu. Lossdetection: Real-time packet loss monitoring system for sampled traffic data. *IEEE Transactions on Network and Service Management*, 20(1):30–45, 2023. DOI: 10.1109/TNSM.2022.3203389.

[34] Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, and Odej Kao. Performance diagnosis in cloud microservices using deep learning. In Hakim Hacid, Fatma Outay, Hye-young Paik, Amira Alloum, Marinella Petrocchi, Mohamed Reda Bouadjenek, Amin Beheshti, Xumin Liu, and Abderrahmane Maaradji, editors, *Service-Oriented Computing – ICSOC 2020 Workshops*, pages 85–96, Cham, 2021. Springer International Publishing. ISBN 978-3-030-76352-7.

[35] Boyi Xian, Weigang Hou, Yue Zong, Jingjing Wu, and Lei Guo. Prediction-based and provident service degradation in optical data center networks. In *2016 15th International Conference on Optical Communications and Networks (ICOCN)*, pages 1–3, 2016. DOI: `10.1109/ICOCN.2016.7875717`.

[36] Jiahao Zhu, Lu Zhao, Jian Zhou, Weidu Ye, and Fu Xiao. Service degradation-tolerated online user allocation in edge computing. *IEEE Transactions on Services Computing*, pages 1–14, 2024. DOI: `10.1109/TSC.2024.3353290`.

[37] Annus Zulfiqar, Ben Pfaff, William Tu, Gianni Antichi, and Muhammad Shahbaz. The slow path needs an accelerator too! *SIGCOMM Comput. Commun. Rev.*, 53(1): 38–47, apr 2023. ISSN 0146-4833. DOI: `10.1145/3594255.3594259`. URL `https://doi.org/10.1145/3594255.3594259`.